Тема 7. Колекции (Контейнери на Java)

Съдържание

- Същност и начин на реализация;
- Интерфейс Колекция (Collection);
- Темплейтизирани колекции (Generics);
- Преобразуване между масиви и колекции;
- Итератор;
- Типове колекции (Интерфейси);
  - Set, SortedSet;
  - List.
  - Map, SortedMap;
- Класовата йерархия на типовете за съхраняване;
- Общи методи на базовия клас Object, използвани от колекциите;
- Имплементиране на сравнение за подредба.

Същност и начин на реализация

Същност и обености на интерфейса колекция (Collection)

- Същност-Абстракция за представяне на група от единични елементи (Object);
- Особености:
  - Форма на представяне -интерфейс;
  - Динамична имплементация-променлива размерност;
  - Сравнение с масиви – разлики;
  - Възможност за представяне на абстрактни отношения между обектите

Същност и начин на реализация

Преди версия 5.0 има два съществени недостатъка:

- Работят с анонимни типове данни – представяне чрез референция към Object – следствия:
  - Типови преобразувания към реалния тип след извличането му от колекцията.
  - Намалява сигурността на типовете по време на изпълнение;
  - Не работят с простите типове – решение чрез заграждащите им класове. Следствие:
    - Непрекъснатото обвиване и разкриване от класовете- производителност и грешки.

Същност и начин на реализация

Подобрения след 5.0 колекциите са изградени на две основни особености:

- Шаблонните типове за типова сигурност на програмите;
- Автоматично обвиване и разкриване от класовете на реалните типове, които са заградени в тях.

Резултат:

- Намаляване на кода, който се пише;
- Увеличава сигурността му;
- Програмистът борави с тези типове без да се интересува от начина на вътрешното съхранение в колекцията.

Същност и начин на реализация

В началните си версии езика има само два класа за представяне на динамични колекции:

- java.util.Vector class – динамичен списък и
- java.util.Hashtable, class – за съхраняване на двойки ключ/стойност

Развитие на система от контейнери в рамка, наречена **Collections Framework**. Старите класове са запазени, но развойните среди дават съобщение за наличието на новите класове.

Същност и начин на реализация

Същност на **Collections Framework**.

- Добавяне на нови абстракции - интерфейси за представяне;
- Добавяне на множество от нови класове, които разширяват възможностите;
- Свързват алгоритми, итератори и колекции;
- Реализират стандартната темплейтна библиотека в езика.

Същност и начин на реализация

Начин на реализация

- В Java се използва подходът на наследяването;
- Контейнерите са разработени да съхраняват референции към **Object - следствия:**
  - Наследник може да се съхранява в контейнер;
  - За да се съхрани обект в контейнер, трябва да е от референтен тип;
  - Контейнерът преобразува типа на обекта към **Object** и той губи своята единтичност:
    - Използва се upcasting
  - Когато се извлече обратно от колекцията, резултатът е референция към **Object,** а не референцията към типа, който е съхранен в контейнера:
    - Използва се обратно преобразуване на типа към наследения тип за да се получи съхранения обект- downcasting.

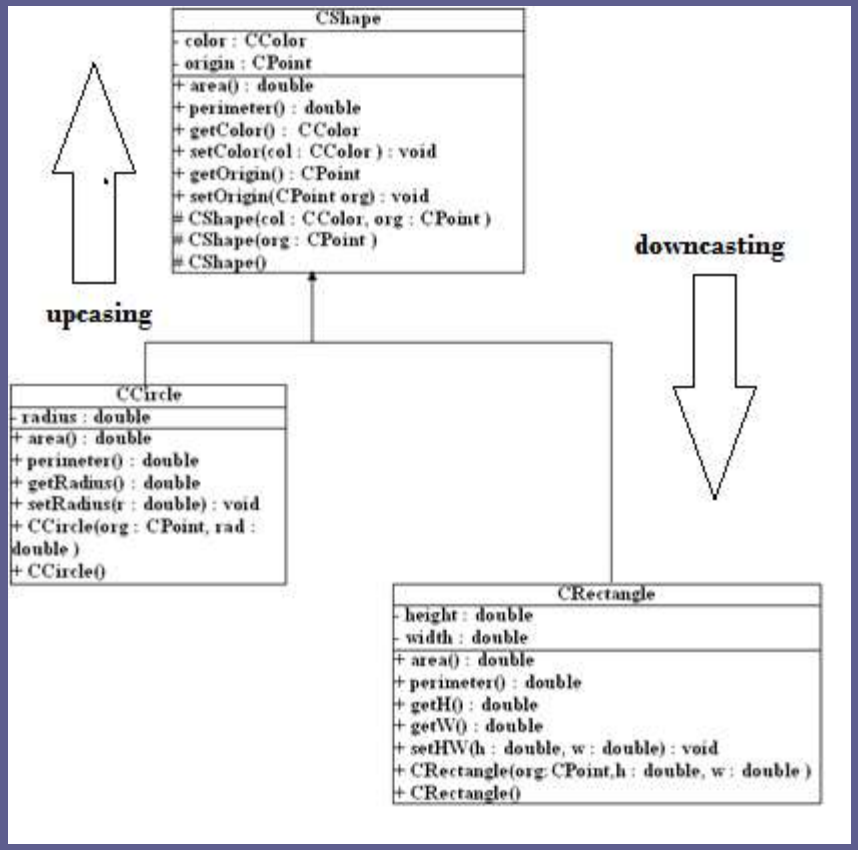
Същност и начин на реализация

Начин на реализация. Основа е наследяването и имплементация на интерфейси. Основни подходи:

- Преобразуване на типа надолу (към базовия клас) – upcasting е известно, че например **CCircle, CRectangle са** наследници на **CShape;**
- При обратното преобразуване (downcasting) не е известно какво точно е съхранено в контейнера. Трябва да се проверява за съответствие:
  - Проверката и преобразуването по време на изпълнение изисква време и внимание при програмиране.

Същност и начин на реализация

Начин на реализация, типови преобразувания- оперират с обекта като обект на другия тип



Видове:

- Възходящо преобразуване на типовете (upcasting) - оперира с обект на производен тип като с обект на базовия тип (движещ се по диаграмата на наследяване);
- Низходящо преобразуване на типовете (downcasting) - оперира с обекта, като представител на конкретния клас

Същност и начин на реализация

Добавяне на параметризираните типове:

- Подобрява решението с използване на параметризираните типове:
  - Компилаторът проверява правилното използване на контейнера-добавяне,извличане;
  - Позволява съхраняването само на обекти по избрания шаблонен тип и контролира операциите с контейнера.

Същност и начин на реализация

Начин на реализация

Разположение на колекциите - Пакет: java.util

Две основни йерархии на колекциите:

- Първата започва **от интерфейса Collection**. Този интерфейс и имплементациите от класове му представя контейнер, който съхранява в себе си други обекти;
- Втората започва **от интерфейса Map**, който представя групата от колекции, основани на двойки ключ/стойност.

Интерфейс Колекция (Collection)

Същност:

Това е абстракция (интерфейс), предназначен за представяне на операциите за съхраняване:

- Не е определен точният начин за организация на данните;
- Не се определя дали има повторяемост, подредени ли са в колекцията и др. Тези детайли на операциите се представят в наследниците на интерфейса.

Интерфейс Колекция (Collection)

Основни методи на интерфейса:

public boolean add(element) – добавяне на елемент в колекцията:

- Връща резултат при успешно добавяне – true;
- Ако обектът съществува в колекцията и тя не допуска дублиране- false;
- Ако колекцията е само за четене и се изпълнява операцията, се генерира изключение UnsupportedOperationException.

Интерфейс Колекция (Collection)

Основни методи на интерфейса:

public boolean remove(element)

- Премахване на елемент от колекцията. Резултат – аналогичен на add. Изключение – UnsupportedOperationException.

public boolean contains(element)

- Връща true ако element се съдържа в колекцията.

public int size( ) - Брой на елементите в колекцията.

public boolean isEmpty( ) - връща true ако колекцията няма елементи.

Интерфейс Колекция (Collection)

Основни методи на интерфейса:

- public Iterator iterator( ) – метод за получаване на обект за достъп до всички елементи на колекцията. Връща обект чрез интерфейса **Iterator:**
  - Обектът служи за преминаване през елементите на колекцията;
  - Достъп до единичните обекти;
  - Премахване на елемент.

Интерфейс Колекция (Collection)

Основни методи на интерфейса:.

- addAll( ), removeAll( ) и containsAll( ) за групови операции с други колекции:
  - Операции добавяне, премахване или проверка за всички елементи на подадената като аргумент колекция;
  - Параметър- абстракция Collection.

Темплейтизирани колекции (Generics)

Същност:

Типът на колекцията се параметризира със специфичния тип на елементите, които ще се съхраняват в нея:

- Преобразува колекцията от съхранител на референтен тип (Object) в съхранител на конкретен тип. Представя се по време на компилация;
- Типът на елементите се контролира във всички методи на колекцията - add( ), remove( ), contains( ) и др.

Темплейтизирани колекции (Generics)

Пример:

ArrayList е един от класовете имплементиращи Collection:

- Тип – Динамичен масив от Object;
- Като се използва параметризация, се контролира съдържанието, което може да се добавя/получава.

Темплейтизирани колекции (Generics)

Например, колекция за тип Date:

```
Collection<Date> dates = new ArrayList<Date>(  );
dates.add( new Date(  ) );
dates.add( "foo" ) // продуцира грешка - не е Date!!
```

При параметризация, методът за добавяне не допуска друго, освен дата.

Темплейтизирани колекции (Generics)

- Ако в примера се използва друг тип- компилационна грешка.
- Без параметризация-тип Object и необходимо преобразуване, например:

```
Collection dates = new ArrayList(  );
dates.add( new Date(  ) );  // остава
// непроверено, предупреждение от компилатора
```

Преобразуване между масиви и колекции

Методите, с които могат да се преобразуват колекциите в масиви са следните:

```
public Object[] toArray(  )
public <E> E[] toArray( E[] a )
// не темплейтен вариант
// public Object[] toArray( Object [] a )
```

- Първият метод връща масив от базовия клас (Object);
- Втората форма е темплейтизирана с тип (клас) E. **Ако съществува масив с необходимия размер, то той ще се запълни с обекти. Но ако е по-малък, то новият масив ще се създаде с необходимият размер и ще се върне като резултат**.

Преобразуване между масиви и колекции

Преобразуването работи и с празен масив:

String [] myStrings = myCollection(new String[0]); // темплейтен

Обратното преобразуване, например от масив към списък, може да се реализира със статичния метод asList (), който е метод на java.util.Arrays:

List list = Arrays.asList(myStrings);

Итератор (Iterator)

Същност:

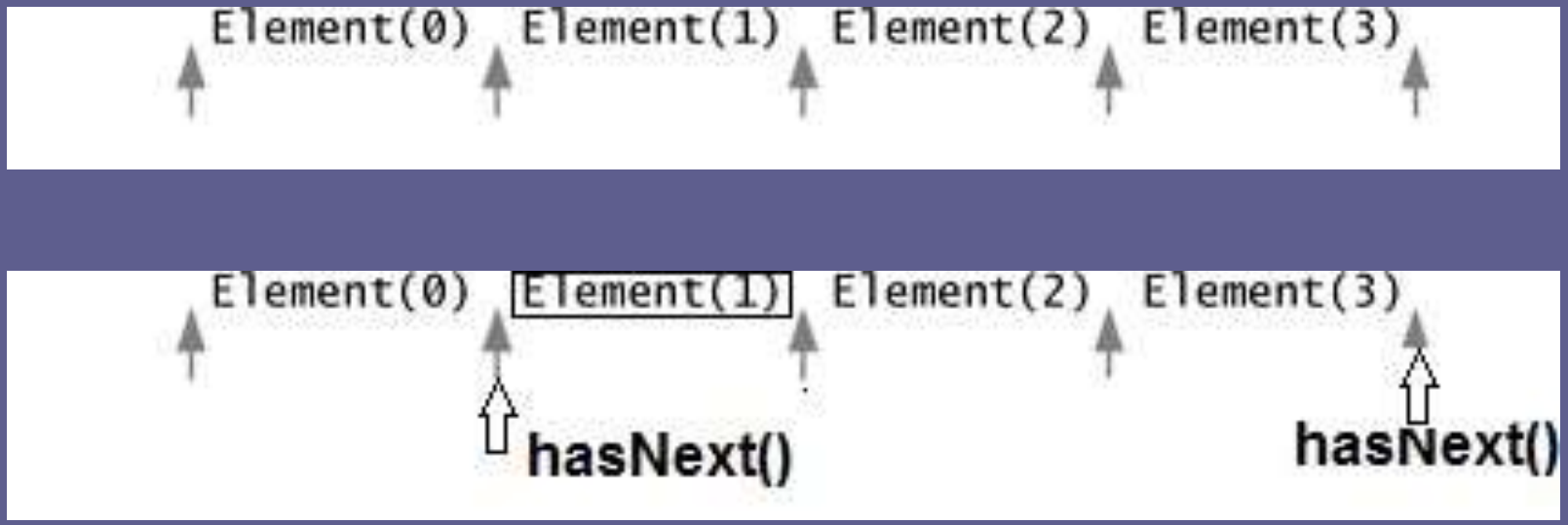
Итераторът е обект, който служи за преминаване през колекциите.

Осигурява чрез методи операциите в собствен стандартизиран интерфейс:

```
Interface Iterator{
    boolean hasNext();
    Object next();
    void remove();
}
```

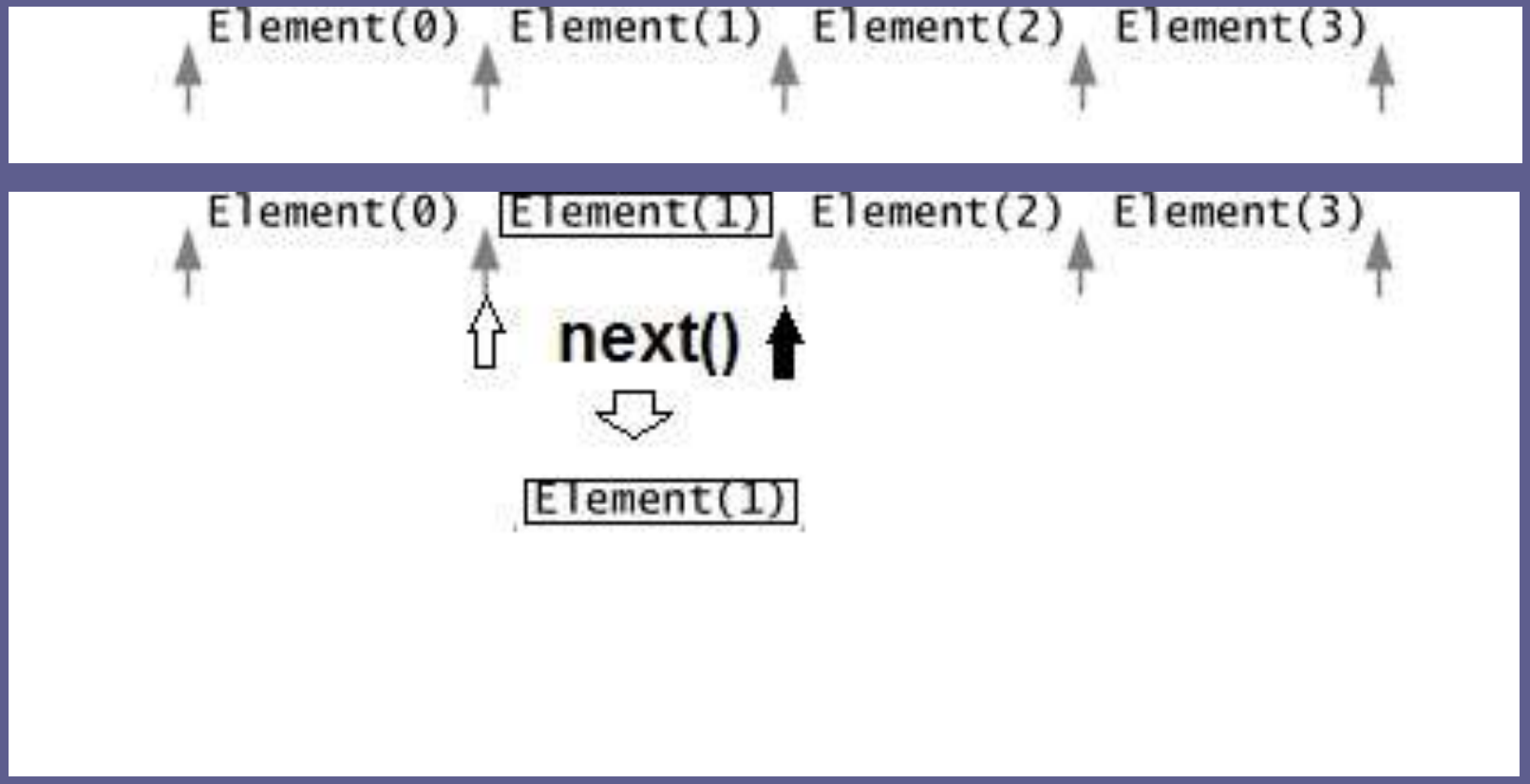
Итератор (Iterator)

Принцип на действие. Метод за проверка hasNext()



Итератор (Iterator)

Принцип на действие:



Итератор (Iterator).Методи

Интерфейсът Iterator има два основни метода:

public Object next( ) – връща **следващия** елемент от колекцията.

public boolean hasNext( ) – връща true, ако може да се премине на следващ елемент за прочитането му. Правило: **ако резултатът на hasNext( ) е true, може да се стартира метода next( ) и да се получи елемента**.

Третият му метод служи за премахване: **remove( ).**

Итератор (Iterator).Методи

Пример за използване на Iterator за извеждане на елементите на една колекция:

```
public void printElements(Collection c, PrintStream out)
{
        Iterator iterator = c.iterator(  );
        while ( iterator.hasNext(  ) )
            out.println( iterator.next(  ) );
}
```

Итератор (Iterator).Методи

public void remove( )

- Премахва последния обект, получен при изпълнението на next( ) от асоциираната колекция;
- Не се предоставя от всички колекции-има колекции само за четене:
  - UnsupportedOperationException
- remove( ) преди изпълнение на next( ) и двукратно:
  - IllegalStateException.

Оператор for за колекции

В Java 5.0 се допуска общия вариант на цикъл за достъп до цялата колекция:

- Пример с операторния вариант на for:

```
Collection<Date> collection = ...
for( Date date : collection )
    System.out.println( date );
```

- Предоставя се само за типа Collection не и за Map:
  - В Map се съхраняват двойки и няма еднозначност на елемента, който се достъпва(ключа или стойността).

Типове колекции

Интерфейси:

- Collection;
  - Set (SortedSet);
  - List.

Типове колекции

Интерфейсът **Collection** има два основни интерфейса-наследници.

- **Set** представя колекция, в която не се допуска дублиране на елементите;
- **List** е колекция, чиито елементи имат определен ред;

Типове колекции. Set

Дефиниция на Set

.

```
public interface Set<E> extends Collection<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);
}
```

Типове колекции. Set

Същност на Set

- Имплементира методите на Collection;
- Осигурява правило за недопускане на дублиране:
  - Ако се направи опит за извикване на метода add( ) при наличие на елемента в колекцията, резултатът е false;
  - **Проверката за дублиране се основава в крайна сметка, независимо от типа на set от (предефинирания) equals**

Типове колекции. SortedSet

Дефиниция:

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

Типове колекции. SortedSet

Особености:

SortedSet добавя няколко метода към Set. При извикване на add( ) и remove( ) се управлява реда на елементите.

Може да се създават:

- Подмножества **subSet( ) по from, to;**
- **Глава headSet( ) по to;**
- **Опашка tailSet( ) по from**.

Методите first( ), last( ) и comparator( ) връщат първи, последен и обектът, използван за сравнение на елементите.

Типове колекции. List

Същност:

List представя линейна колекция, подобна на масив с методи за промяна на позицията на елементите. Методи:

public void add(E *element*) добавя елемент в края на списъка;

public void add(int *index*, E *element*) вмъква елемент на позицията, която е параметър. Предизвиква изключение IndexOutOfBoundsException по невалиден ***index***. Следващите елементи се изместват с позиция след вмъкнатия на индекса index.

Типове колекции. List

Дефиниция:

```
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);
}
```

Типове колекции. List

public void remove(int *index*) премахва елемента на указаната позиция;

public E get(int *index*) връща елемента на указаната позиция;

public Object set(int *index*, E *element*) промяна на елемента на указаната позиция със специфицирания като параметър. Предизвиква IndexOutOfBoundsException.

Типът на E в методите е референция към имплементирания от класа List тип.

Интерфейс Map

Дефиниция:

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

    // Basic operations
    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();

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

    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Интерфейс Map

Пакет: java.util.Map

Същност: Това са изветните съхранители “речник” или “асоциативен масив”:

- Поддържат и достъпват елементите чрез ключовата стойност;
- При запис на елемент в контейнера се асоциира записаната стойност към ключа.
- При темплейтизирания вариант Map се създава с два параметризирани типа-за ключа и за стойността.

Интерфейс Map

- Методи:

public V put(K *key*, V *value*) - добавяне на двойка ключ/стойност;

public V get(K *key*) получаване на стойност по ключ;

public V remove(K *key*) премахване на стойност, по ключа;

public int size( ) – брой на двойките ключ/стойност;

public Set keySet( ) – връща в Set на всички ключове;

public Collection values( ) – връща в Collection всички стойности на обекта (може да има дублирани елементи);

Интерфейс Map

Пример:

```
Map<String, Date> dateMap =
                    new HashMap<String, Date>(  );
dateMap.put( "today", new Date(  ) );
Date today = dateMap.get( "today" );

// непараметризиран вариант
Map dateMap = new HashMap(  );
dateMap.put( "today", new Date(  ) );
Date today = (Date)dateMap.get( "today" );
```

Непараметризираният вариант подразбира тип Object и изисква типово преобразуване при достъпа

Интерфейс SortedMap

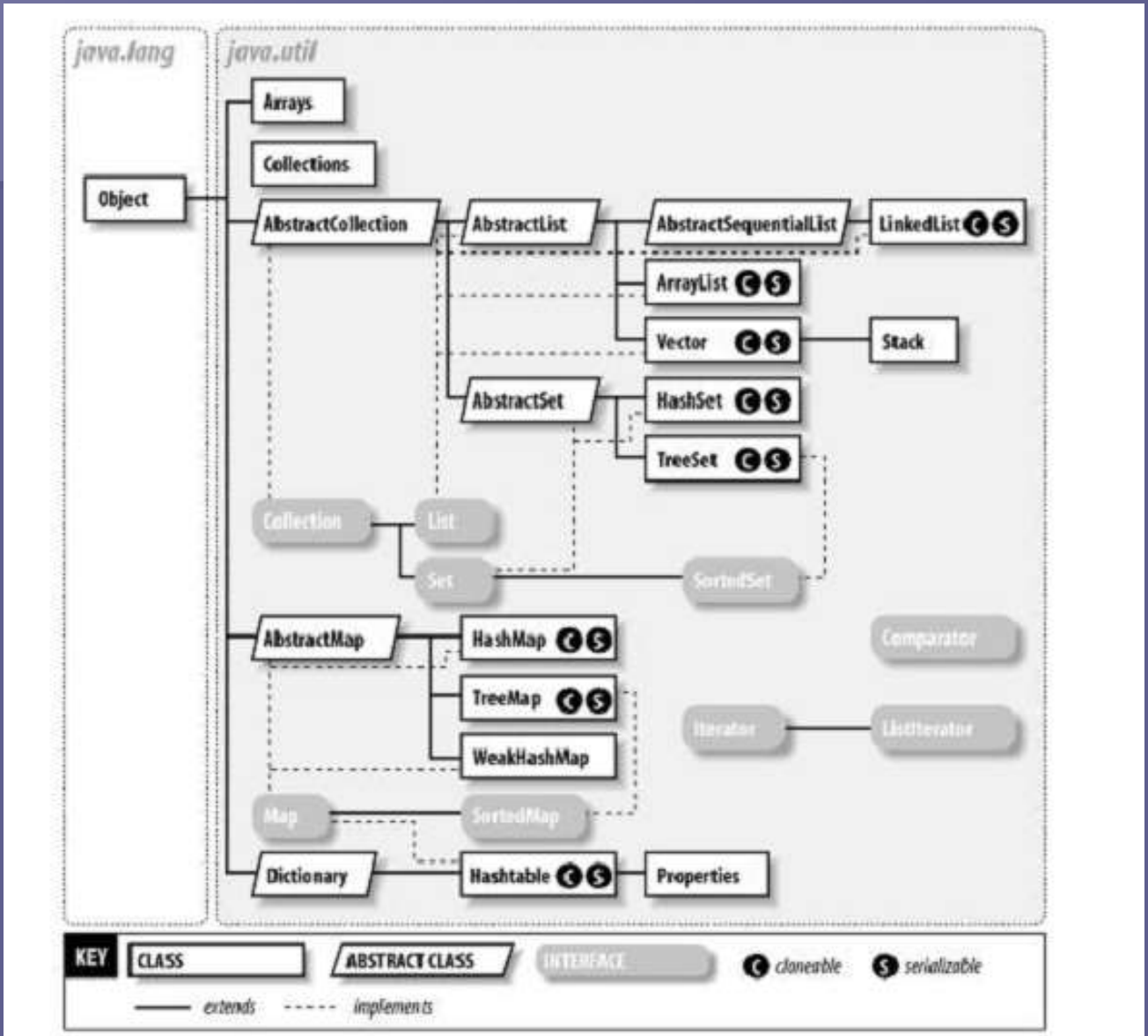
Дефиниция на SortedMap.

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

    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();

    Comparator<? super K> comparator();
}
```

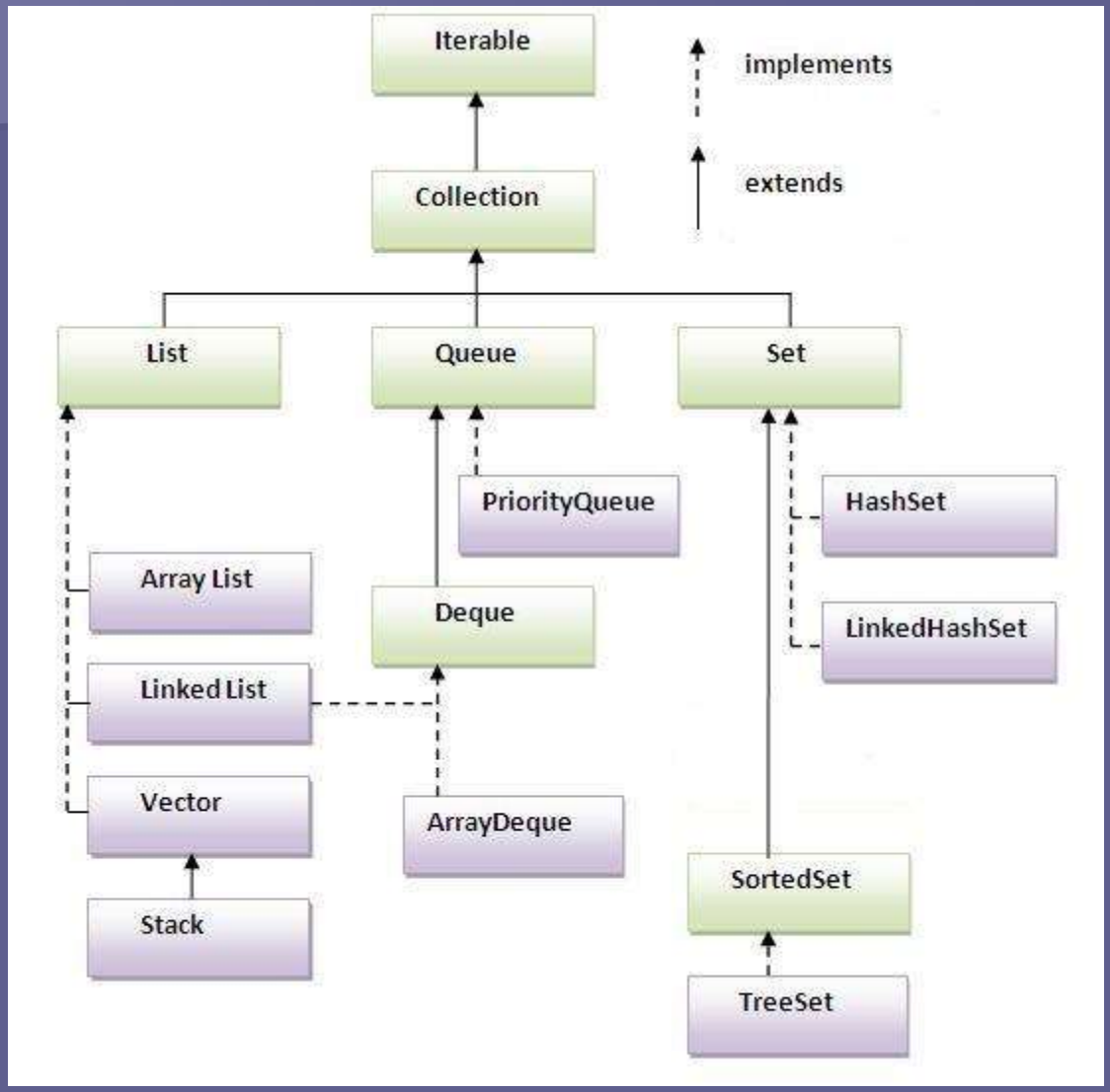
Интерфейс Map, SortedMap

- SortedMap наследява интерфейса Map като:
  - Елементите са сортирани по реда на ключовата стойност;
  - Предоставя методите subMap( ), headMap( ) и tailMap( ) за получаване на подмножества от сортирания map;
  - Предоставя метод comparator( ) който връща сравнителя за сортиране.
- Има два отделни вида итератора – за ключа и за стойността.

Класовата йерархия на типовете за съхраняване



Класовата йерархия на типовете за съхраняване разширена



Класовата йерархия на типовете за съхраняване

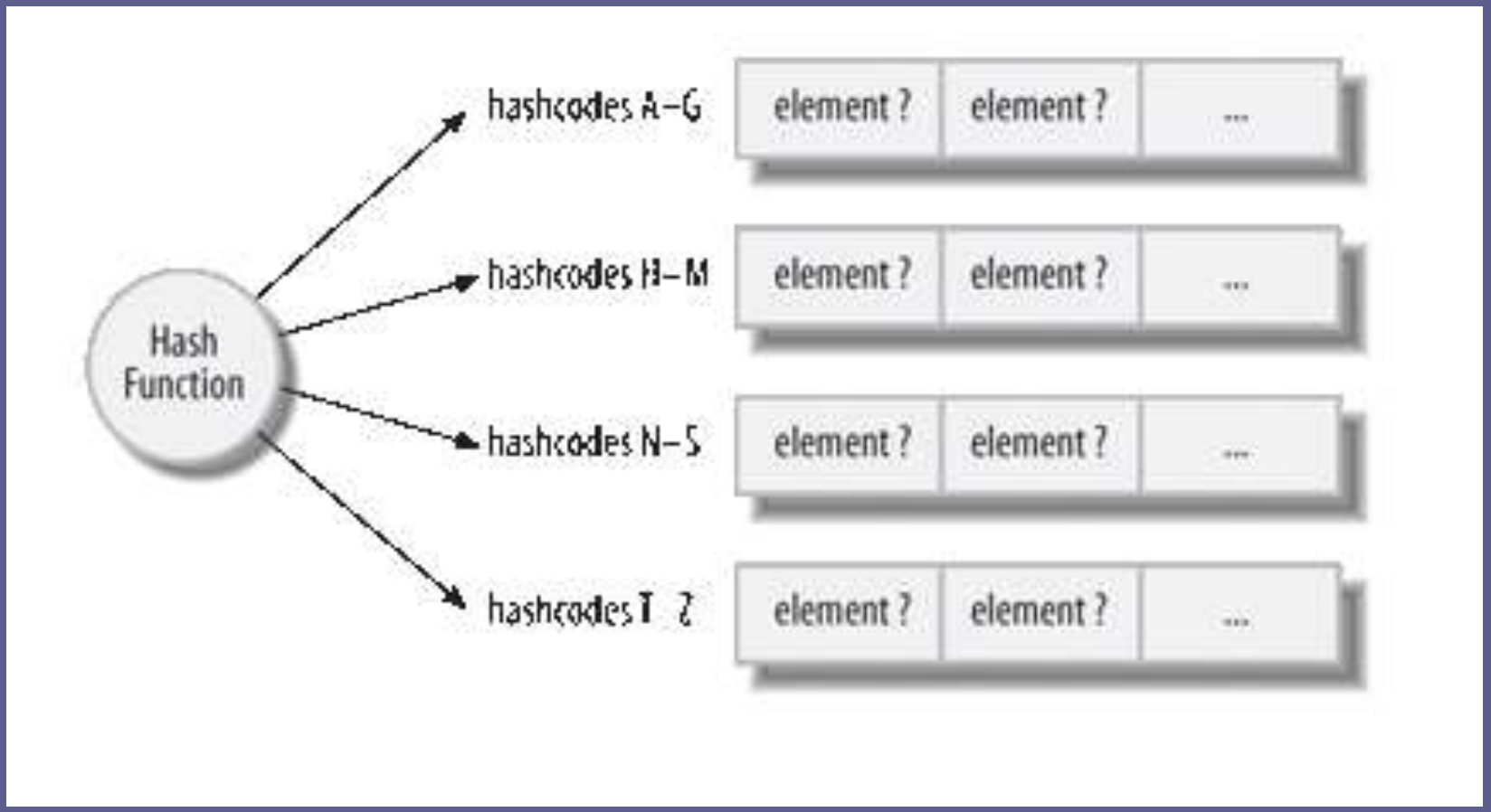
- Използването на контейнерите на езика за съхраняване на обекти от произволен клас се основава на класовата йерархия с базов клас Object.
- **За да се осигури организация на обектите в колекциите се използват методите на базовия клас Object. Те се предефинират в класовете за съхраняване в колекции и осигуряват съответните методи за достъп на колекцията.**

Общи методи на базовия клас Object, използвани от колекциите

- **int hashCode()** връща hash кода на обекта; Методът се използва за получаване на кода необходим при запис на обектите **в хеширани таблици**.

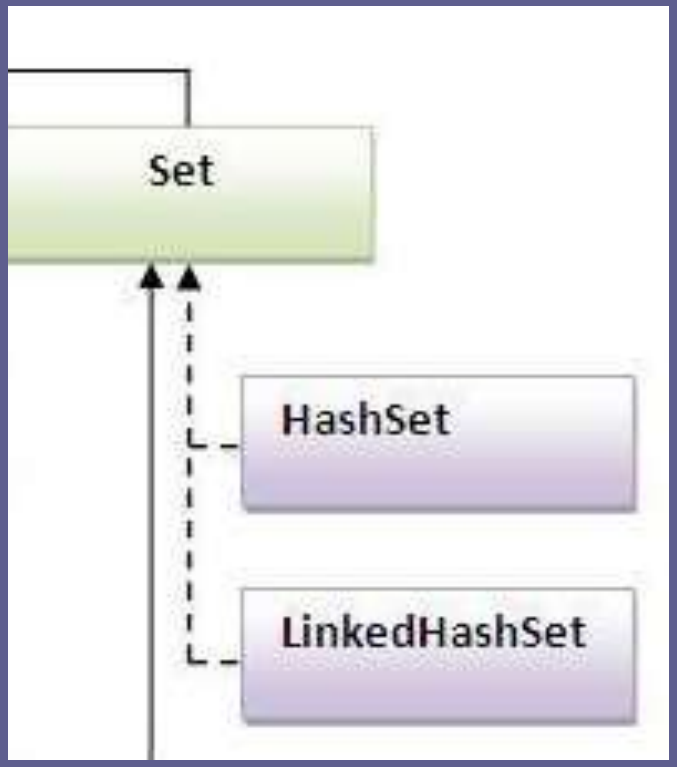
Общи методи на базовия клас Object, използвани от колекциите

- Организация на хеширащи таблици:



Общи методи на базовия клас Object, използвани от колекциите

- Класове, основани на хеширане:



Общи методи на базовия клас Object, използвани от колекциите

- **Основни принципи при имплементация на hashCode():**
  - да се обединят **всички стойности на данни, които се различават,** в хеш код;
  - **битовете на данните, които се променят най-много** да засягат **по-младшите битове** на хеш кода;
  - възможно повече битове да имат случайно разпределение (0, 1), особено младшите;
  - висока скорост-прости операции;

Общи методи на базовия клас Object, използвани от колекциите

Примерни имплементации на hashCode() при два еднакви елемента x,y:

- **int :**
  - с операция XOR - x ^ y при "много" равномерно разпределени;
  - (x * p) ^ y, където p е просто или "малко", нечетно, около степените на 2 число– напр. 17 или 33.
- **Класове:** x.hashCode() ^ y.hashCode();
- **double:**
  - new Float(x).hashCode() ^ new Double(y).hashCode()
  - (new Double(x).hashCode() >> 13) ^ new Double(y).hashCode()
- String: hash(s) = $s_0*31^{n-1} + s_1*31^{n-2} + \ldots + s_n$ n - дължина

Общи методи на базовия клас Object, използвани от колекциите

Примерни имплементации на hashCode() при различни полета, елементи в клас:

- **boolean**, за **true** се представя **1**, **false** с **0**.
- **int**, **byte**, **short**, **char** - преобразуване в **int**, чрез (**int**);
- **long**, разделя се на 2 по 32 бита като **int** стойности;
- **float** (**double**), превръщане в целочислен вид чрез методите **Float.floatToIntBits()** или **Double.doubleToLongBits()**. Резултатът **double** се третира както **long**;
- Референтен тип - извикване метода **hashCode()**;.
- Масив или колекция, извлича се хеш-кода за всеки елемент на колекцията.

Общи методи на базовия клас Object, използвани от колекциите

boolean equals(Object toCompare)

- Служи за дефиниране на оператор за сравняване на обектите.
- Може (трябва) да се предефинира в потребителския клас с цел да се използва операцията при съхраняването им в контейнерите и търсене на обекти;
- Тъй като HashSet по-специално се основава на хеш таблица, той ще търси дублиране чрез изчисляване на hashCode() на представения му обект и след това ще премине през всички обекти, ако има такива, в съответната група с еднакъв код.

Общи методи на базовия клас Object, използвани от колекциите

Изисквания към функцията, реализираща метода **equals**:

- рефлексивна - за всяка референция x x.equals(x) е винаги истина;
- симетрична- за всяка референция x и y, x.equals(y) е true тогава и само тогава, когато y.equals(x) е true;
- транзитивана - за всяка референция x, y и z, ако x.equals(y) е true и y.equals(z) е true, тогава x.equals(z) е true.
- консистентна – една функция е консистентна, ако за всяка референция на обектите-x и y многократните извиквания на x.equals(y) винаги връща един и същ резултат от сравнението;
- сравнение с нулева референция. За всяка референция към обект obj, която е ненулева obj.equals(null) е винаги false

Общи методи на базовия клас Object, използвани от колекциите

Правило, което трябва да се изпълнява за осигуряване на консистентност между функциите, кодиращи **hashCode() и equals():**

Ако x.equals(y) връща true

=>

x.hashCode() == y.hashCode()

(хеш кодовете на x и y трябва да са равни)

Общи методи на базовия клас Object, използвани от колекциите

Пример 1 – кодиране на hashCode и equals за CPoint :

```
public class CPoint {
private int x;
private int y;
// код на конструктори
 public int hashCode() { return (x * 31) ^ y; }
 public boolean equals(Object obj) {
  if (obj instanceof CPoint) {
    CPoint other = (CPoint) obj;
    return (x == other.x && y == other.y);
  }
  return false;
 }
}
```

Общи методи на базовия клас Object, използвани от колекциите

Пример 2: Предефиниране на сравнение за равенство между обектите от класа Person, която имплементира равенството на всички стрингови атрибути. Добавяне на метод **boolean** equals(Object obj) за сравнение:

Общи методи на базовия клас Object

```
public boolean equals(Object obj) {
    if (this == obj)        return true;
    if (obj == null)        return false;
    if (getClass() != obj.getClass()) return false;
    final Person other = (Person) obj;
    if (strName == null) {
            if (other.strName != null) return false;
    } else if (!strName.equals(other.strName)) return false;
    if (strSex == null) {
            if (other.strSex != null)
                    return false;
    } else if (!strSex.equals(other.strSex))
            return false;
    if (strTitle == null) {
            if (other.strTitle != null)
                    return false;
    } else if (!strTitle.equals(other.strTitle)) return false;
    return true;
}
```

Имплементиране на сравнение за подредба

В дефиницията на класа, който ще се съхранява **в сортирана колекция** трябва да се обяви интерфейса, с който да се сравняват обектите при подредбата им в контейнера. Това става с обявяването като наследник на класа интерфейса **Comparable:**

Пример:

```
public class Student extends Person implements Comparable
```

Функцията от този интерфейс, която трябва да се предефинира е **compareTo: compareTo е с резултат int и има смисъла на сравнител, връщащ -1 (по-малко), 1 (по-голямо) и 0 (равно).**

Имплементиране на сравнение за подредба

Пример за предефиниране за сравняване на студенти по факултетен номер в нарастващ и намаляващ ред. Използва се сравнението на тип String, което е организирано на същия принцип. Добавят се методи за сравнението:

Примери.

```
public int compareTo(Object o) {
  return
   (strFacNumer.compareTo(((Student)o).strFacNu
   mer));
}
```

Имплементиране на сравнение за подредба

- За подредба в нарастващ ред може да се използва обратното сравнение:

```
public int compareTo(Object obj) {
  return
   ((Student)obj).strFacNumer.compareTo(str
   FacNumer);
}
```

Колекции на Java

Въпроси?

Следва продължение.....