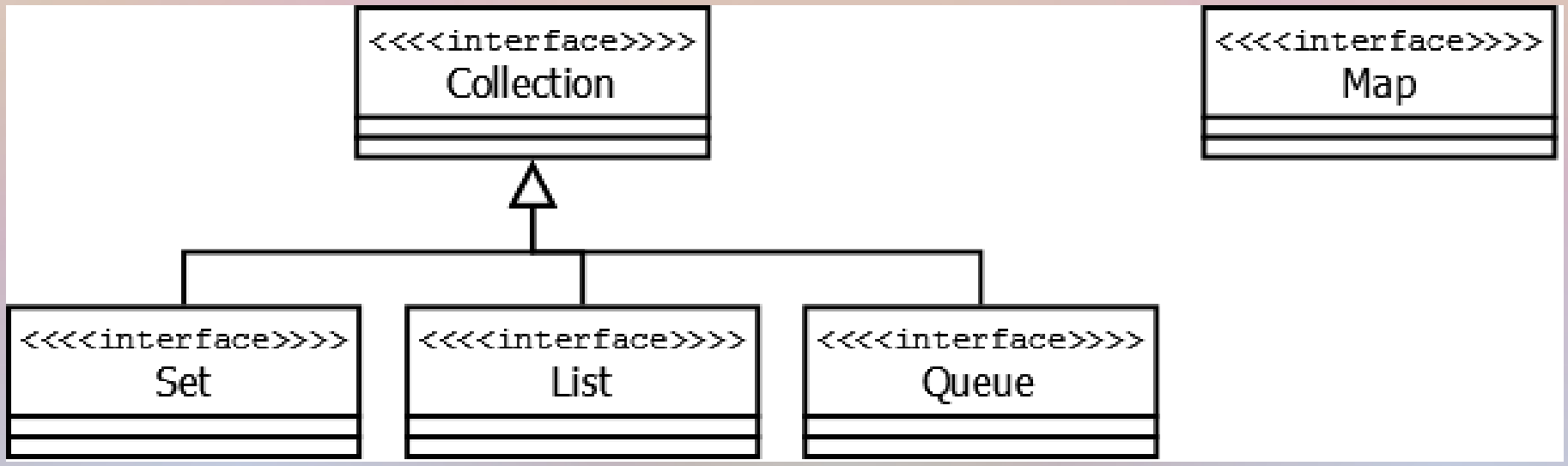
Контейнеры

- Массивы
- Коллекции
- Ассоциативные массивы



Коллекции

- Interface Iterator<E>
  - boolean hasNext()
  - E next()
  - void remove()
- Interface Collection<E>

Перебор элементов коллекции

```
ArrayList <MyClass> collection =  new
                                  ArrayList<MyClass>();
for (Iterator<MyClass>  it = collection.iterator();
                                  it.hasNext(); )
    System.out.println(it.next());
```

Перебор элементов (2)

```
Collection <String> c = . . . ;
Iterator <String> it = c.iterator( ) ;
while ( it.hasNext ( ) ) {
   String element = it.next( ) ;
   // Обработка элемента
}
```

Перебор элементов коллекции в стиле foreach

```
for(MyClass e : collection)
    System.out.println(e);
```

Удаление элемента

```
Collection <Point> points = . . . ;
Iterator <Point > it = points . iterator( ) ;
while ( i t . hasNext ( ) ) {
    Point p = i t . next ( ) ;
    if ( p.equals (new Point(0,0) ) )
    it.remove( ) ;
}
```

Interface Collection<E>

Iterator<E> iterator()

int size()

boolean isEmpty()

boolean contains(Object obj)

boolean containsAll(Collection<?> other)

boolean add(Object element)

boolean addAll(Collection<? extends E> other)

boolean remove(Object obj)

boolean removeAll(Collection<?> other)

void clear()

boolean retainAll(Collection<?> other)

Списки и динамические массивы

- Interface List<E>
- Interface ListIterator<E>
- Class LinkedList<E>
- Class ArrayList<E>

Interface List<E>

ListIterator<E> listIterator()

ListIterator<E> listIterator(int index)

void add(int i, E element)

void addAll(int i, Collection<? extends E> elements)

E remove(int i)

E get(int i)

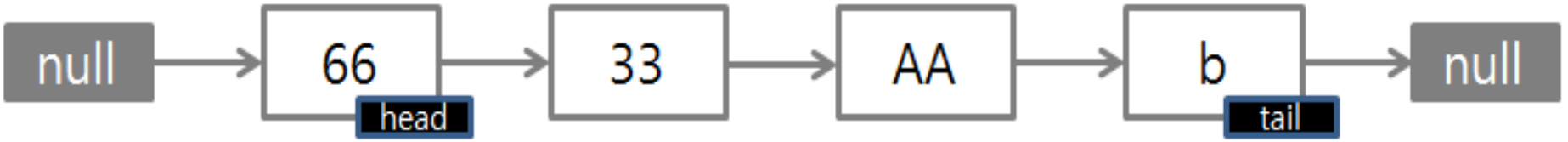
E set(int i, E element)

int indexOf(Object element)

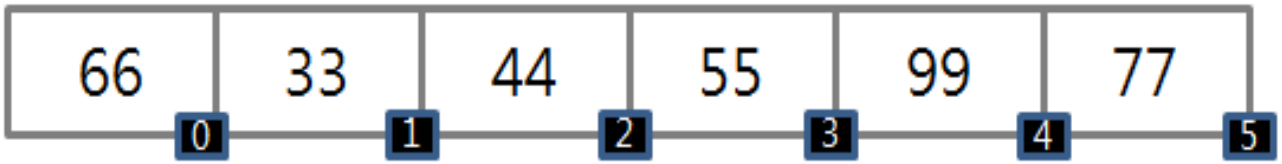
int lastIndexOf(Object element)

Примеры последовательных коллекций

Linked List



Array List



LinkedList

- Двусвязный список
- Быстрая вставка элемента
- Быстрое удаление элемента
- Медленный доступ к произвольному элементу

ArrayList

- Массив
- Медленная вставка элемента
- Медленное удаление элемента
- Быстрый доступ к произвольному элементу

Обход в обратном порядке

```
ListIterator <Point> it =
                    points . listIterator (points.size());
while ( it.hasPrevious( ) ) {
    Point p = it.previous( ) ;
    if (p.getY( ) < 0 ) {
    it.set(new Point (p.getX(), −p.getY())) ;
    }
}
```

Множества и упорядоченные множества

- Interface Set<E>
- Class HashSet<E>
- Interface SortedSet<E>
- Class TreeSet<E>

Interface Set<E>

boolean add(E e)

boolean addAll(Collection<? extends E> c)

void clear()

boolean contains(Object o)

boolean containsAll(Collection<?> c)

boolean isEmpty()

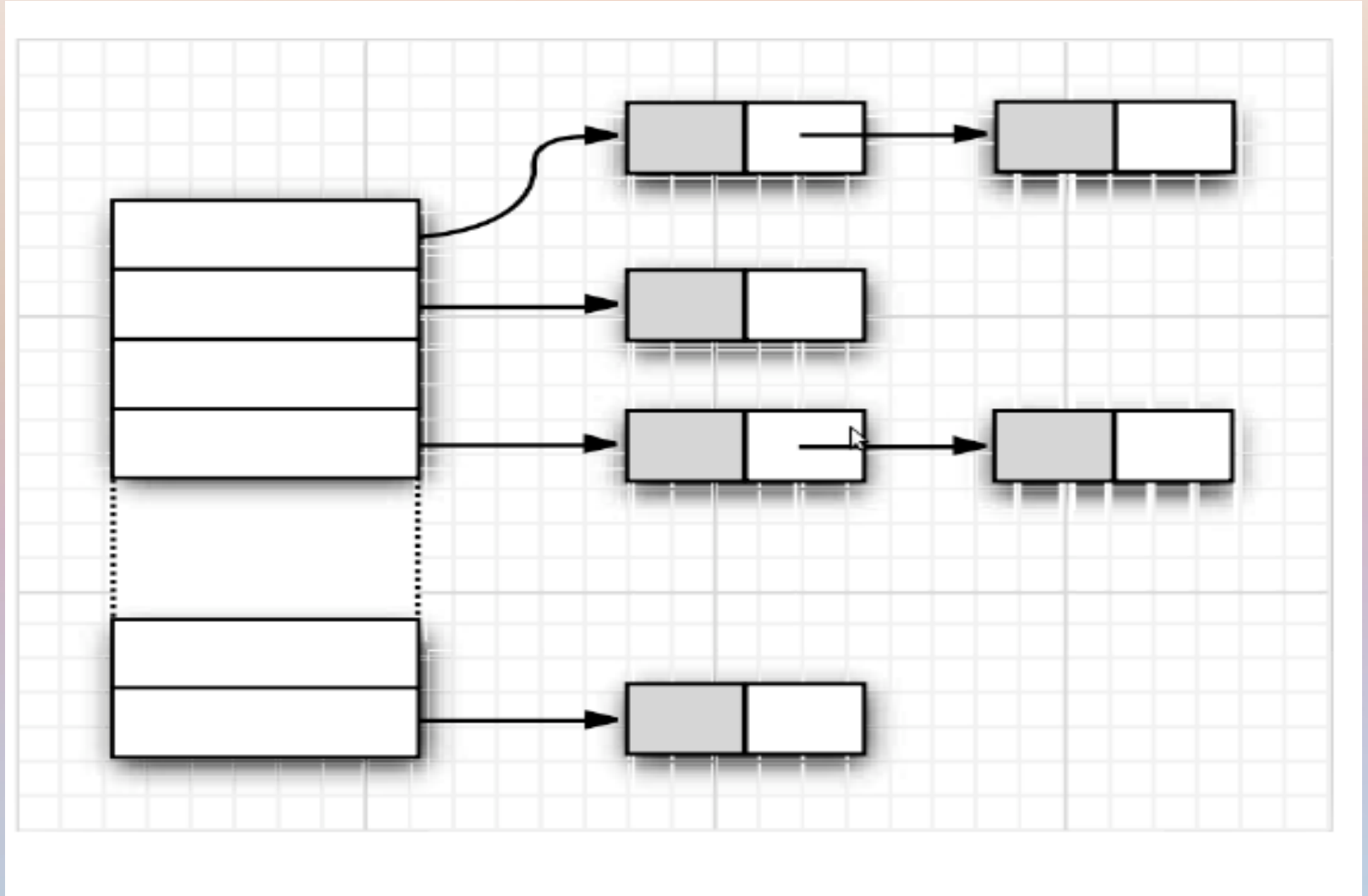
boolean remove(Object o)

int size()

HashSet

- Представляет собой неупорядоченную коллекцию, которая не может содержать дублирующиеся данные. Является программной моделью математического понятия «множество».
- Для хранения объектов используется хеширование (хеш-таблицу)
- Объекты должны реализовывать методы hashCode() и equals()
- Быстрые добавление, удаление и проверка вхождения элемента в множество (не зависят от количества хранимых элементов)

Class HashSet<E>



Реализация методов equals() и hashCode()

- реализуются как переопределение унаследованных методов boolean equals(Object obj) и int hashCode() класса Object;
- если поля объектов равны, то и объекты равны;
- если два объекта тождественны в соответствии с методом equals(), они должны иметь одно значение hashCode() (обратное не верно);
- наличие грамотной реализации equals() и hashCode() объектом позволяет хранить и извлекать их с помощью коллекций на базе хеширования.

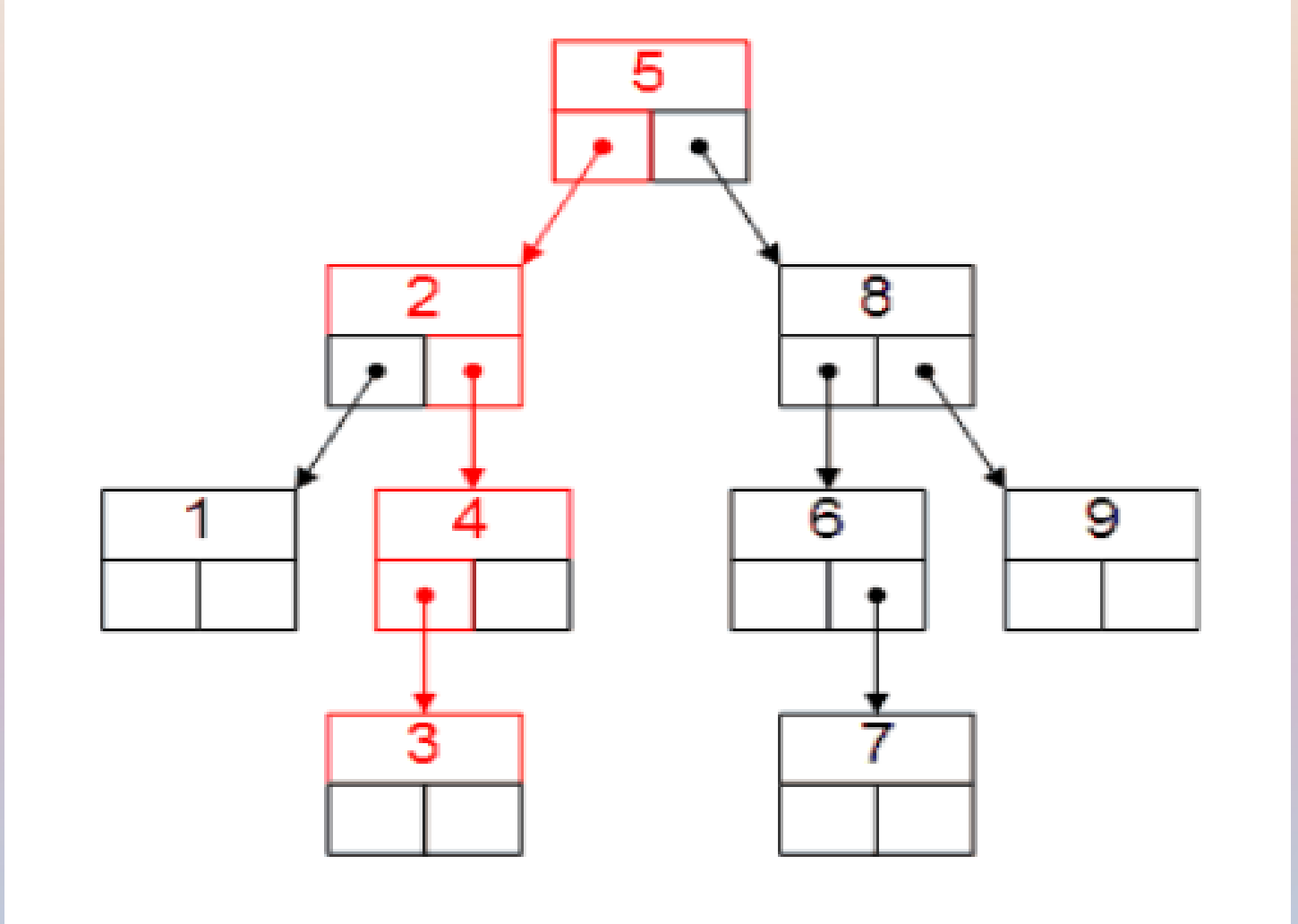
Хеш-код

- хеш-код — это целочисленный результат работы метода, которому в качестве входного параметра передан объект;
- для одного и того-же объекта, хеш-код всегда будет одинаковым;
- если объекты одинаковые, то и хеш-коды одинаковые (в Java по умолчанию не выполняется)
- если хеш-коды равны, то объекты не всегда равны (коллизия);
- если хеш-коды разные, то объекты гарантированно разные.

TreeSet

- Для хранения объектов используется сбалансированное дерево
- Объекты должны реализовывать интерфейс Comparable<E>
- Гарантируется обход итератором в порядке возрастания
- Добавление, удаление и проверка вхождения элемента в множество за время порядка log(N)

Class TreeSet<E>



Упорядочение объектов

- Естественное упорядочение – реализация интерфейса Comparable<E>
  - int compareTo(E obj)
- Внешнее упорядочение – определении внешнего класса компаратора, реализующего интерфейс Comparator<E>
  - int compare(E o1, E o2)

Реализация алгоритмов обработки данных

Класс Collections содержит статические методы для работы с коллекциями данных:

- поиск элементов, максимальных, минимальных;
- сортировка;
- замена элементов.

Некоторые методы Collections

```
void sort(List<T> list)
void copy(List<? super T> to, List<T> from)
void fill(List<? super T> l, T value)
boolean addAll(Collection<? super T> c, T... values)
boolean replaceAll(List<T> l, T oldValue, T newValue)
int indexOfSubList(List<?> l, List<?> s)
int lastIndexOfSubList(List<?> l, List<?> s)
void swap(List<?> l, int i, int j)
void reverse(List<?> l)
void rotate(List<?> l, int d)
```

Ассоциативные массивы

- Interface Map<K,V> (вместо Collection<E>)
- Хранение данных парами – (уникальный ключ, объект)
- Для обхода используются итераторы их ключей
- Class HashMap<K,V>
- Class TreeMap<K,V>

Создание и заполнение ассоциативного массива

```
Map<String, Person > persons =
                                    new HashMap < >();
persons.put("001", new Person("Иванов", 20)) ;
persons.put("002", new Person("Петров", 20)) ;
persons.put("042", new Person("Сидоров", 20)) ;
System.out.println(persons) ;
```

Перебор элементов

```
for(String id: persons.keySet()) {// ...}
for(Person p: persons.values()) {// …}

for(Map.Entry<String, Person> entry: persons.entrySet()) {
String key = entry.getKey() ;
Person p = entry.getValue() ;
// ...
}
```