

A background image showing a complex network of interconnected nodes and lines, resembling a web or a data network, set against a blue gradient.

Стандартная библиотека классов. Generics.

NetCracker®

- Comparator<T>
- Enumeration<E>
- Iterator<E>
- EventListener
- Formattable
- Observer
- *Коллекции*

- **Arrays**
- BitSet
- Calendar, GregorianCalendar
- **Collections**
- Currency
- Date
- Formatter
- Locale
- Observable
- Random
- Scanner

Класс `Date` – класс для хранения даты и сравнения дат.

Классы `Calendar` и `GregorianCalendar` – предоставляет более развитые средства для работы с датой. Методы:

```
get(int field),  
set(int field, int value),  
add(int field, int amount),  
roll(int field, int amount)
```

Класс `Locale` - предназначен для отображения определенного региона.

Пример использования:

```
Locale l = new Locale("ru", "RU");  
Locale l = new Locale("en", "US", "WINDOWS");
```

```
class GenericList<E>
{
    E getFirst() { ... }
    void add(E obj) { ... }
}
```

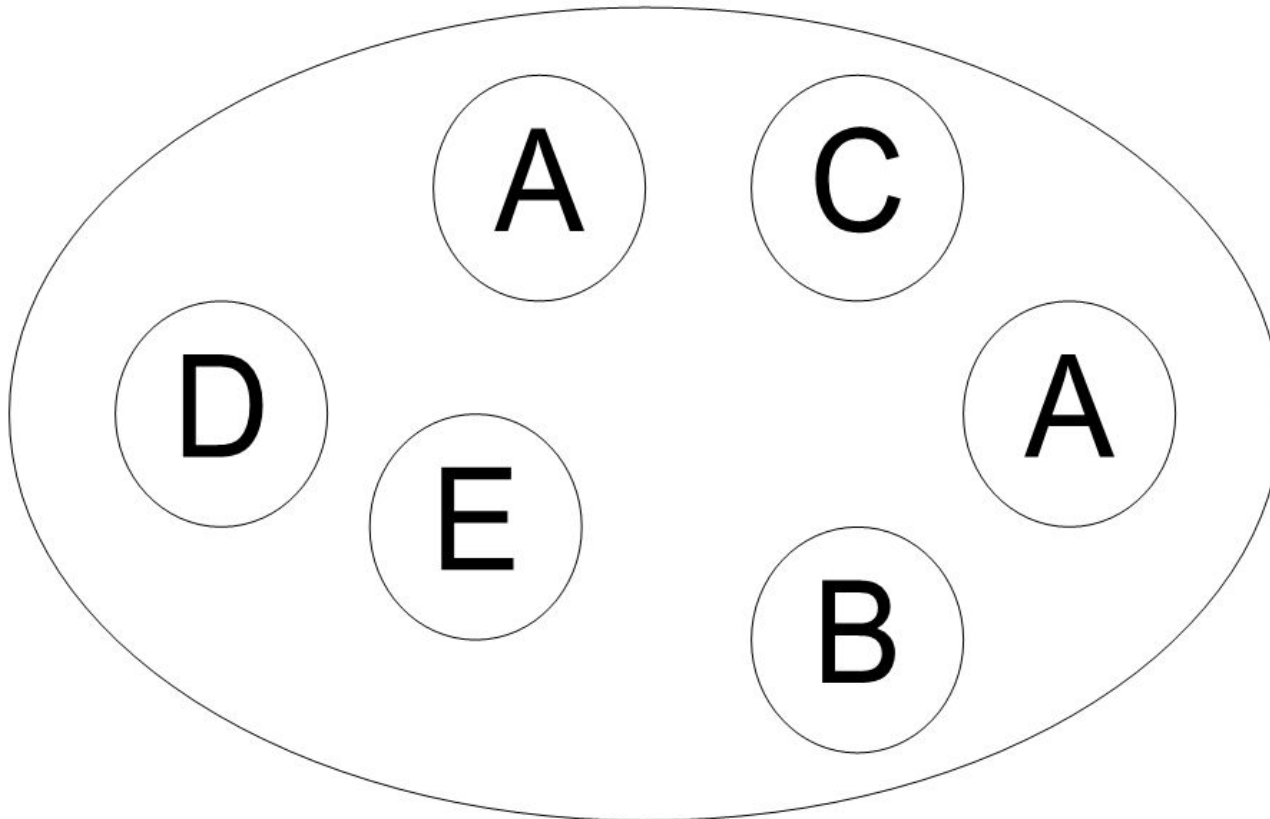
```
class List {
    public Object get(int index);
    public void set(int index, Object value);
}
List list = new List(3);
list.set(0, "string");
list.set(1, new Integer(5));
String value = (String) list.get(0);
```

```
class List<Type> {
    public Type get(int index);
    public void set(int index, Type value);
}
List<String> list = new List<String>(3);
list.set(0, "string");
String value = list.get(0);
```

```
class Util {
    void <T> max(T a, T b);
    double <T extends Number & Cloneable> sum(T a, T b);
    int size(Collection<?> collection);
    void compare(Comparator<? super ArrayList> comparator);
}
Util.<Integer>max(new Integer(1), new Integer(2));
```

```
class Processor <T extends Throwable> {
    void process() throws T; // ok
    void doWork() {
        try {
            process();
        }
        catch (T e) { // ошибка времени компиляции
        }
    }
}
```

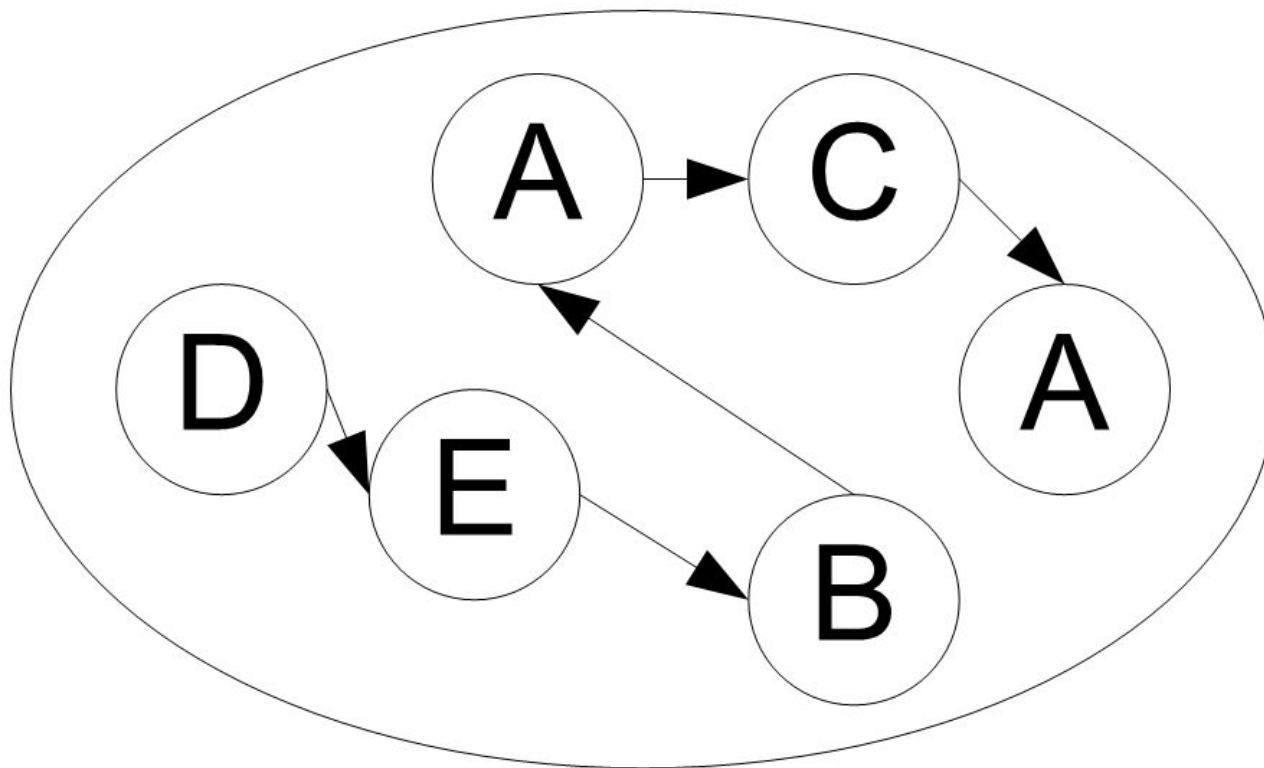
- Коллекция — неупорядоченный набор элементов
- Интерфейс **Collection**



- Определение размера
 - `size()` — количество элементов
 - `isEmpty()` — проверка на пустоту
- Проверки на входжение
 - `contains(Object o)` — одного элемента
 - `containsAll(Collection c)` — всех элементов коллекции `c`

- **Добавление элементов**
 - `add(Object e)` — одного элемента
 - `addAll(Collection c)` — элементов коллекции
- **Удаление элементов**
 - `remove(Object e)` — одного элемента
 - `removeAll(Collection c)` — элементов коллекции
 - `retainAll(Collection c)` — удаление элементов не из коллекции
 - `clear()` — удаление всех элементов
- **Исключения**
 - `UnsupportedOperationException`

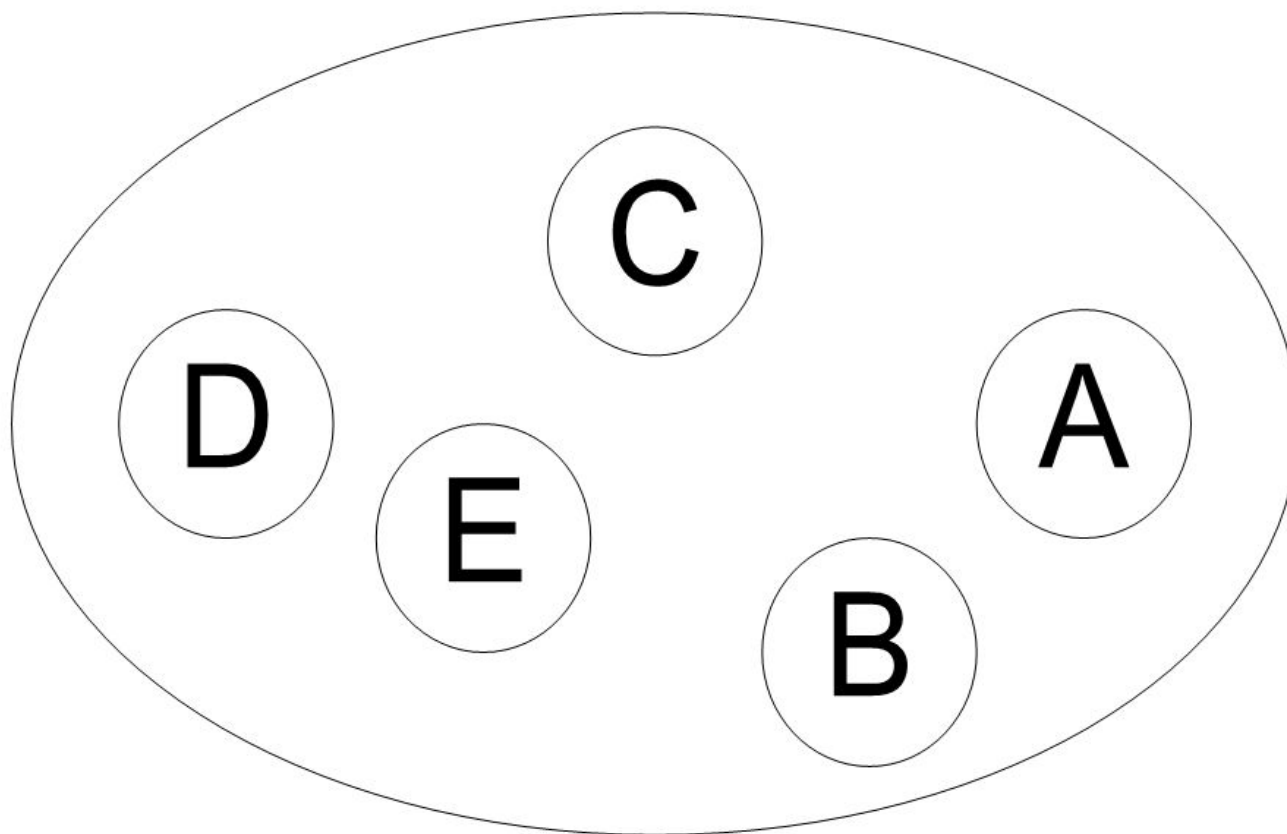
- Итератор — обход коллекции
- Интерфейс `Iterator`
- Метод `Iterator Collection.iterator()`



- `hasNext()` — определение наличия следующего элемента
- `next()` — взятие следующего элемента
- `remove()` — удаление элемента

- Исключения
 - `NoSuchElementException` — бросается при достижении конца коллекции
 - `ConcurrentModificationException` — бросается при изменении коллекции

- Множество — коллекция без повторяющихся элементов
- Интерфейс **Set**

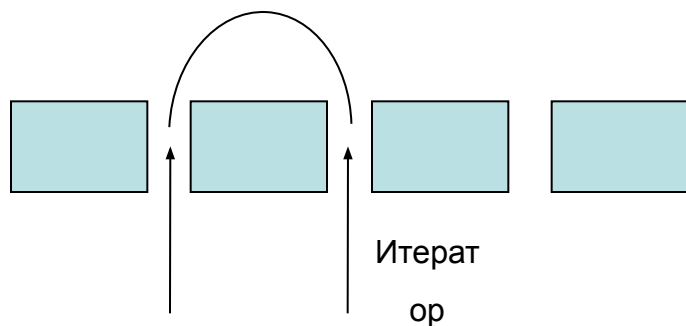


- `addAll(Collection c)` – объединение
МНОЖЕСТВ
- `retainAll(Collection c)` – пересечение
МНОЖЕСТВ
- `containsAll(Collection c)` – проверка
ВХОЖДЕНИЯ
- `removeAll(Collection c)` – разность
МНОЖЕСТВ

- **HashSet** — множество на основе хэша
- **LinkedHashSet** — множество на основе хэша с сохранение порядка обхода

- `HashSet()` — пустое множество
- `HashSet(Collection c)` — элементы коллекции
- `HashSet(int initialCapacity)` — начальная вместимость
- `HashSet(int initialCapacity, double loadFactor)` — начальная вместимость и степень заполнения

- Коллекции (**Collection**) – коллекция элементов, добавление, удаление, размер, очистка, проверка вхождения, перечисление.
 - Списки (**List**) – упорядоченные коллекции, вставка, удаление, чтение по индексу, поиск индекса.
 - *LinkedList, ArrayList, Vector, Stack*
 - Очереди (**Queue**) – коллекции элементов, в которые можно класть и из которых можно брать элементы (коллекции с приоритетами, стеки и очереди)
 - *LinkedList, PriorityQueue*
 - Множества (**Set**) – коллекции элементов, в которые каждый элемент может входить только один раз. Могут быть отсортированными (**SortedSet**).
 - *EnumSet, HashSet, LinkedHashSet, TreeSet*
- Таблицы (**Map**) – сопоставление некоторого значения некоторому уникальному ключу. Таблицы, сортированные по ключу (**SortedMap**).
 - *EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, Properties, TreeMap*



```

Iterator<T>
• boolean hasNext();
• T next();
• void remove();

```

```

Iterable<T>
• Iterator<T> iterator();

```

Единственный способ обращения к элементу основан на вызове его метода `next()`

Метод `void remove()` удаляет объект из коллекции:

- при удалении элемента по заданной позиции необходимо "пройти" его:

```

Iterator it = c.iterator();
it.remove();

```

```

Iterator it = c.iterator();
it.next();
it.remove();

```

```

Iterable<Double> numbers = new ArrayList<Double>();

```

```

for (Double number : numbers)

```

```

    System.out.println(number);

```

Все методы статические.

Обеспечивает набор методов для выполнения операций над массивами.

Методы:

сортировка (18)

```
static void sort(int[] a)
```

бинарный поиск (9)

```
static int binarySearch(int[] a, int element)
```

заполнение (18)

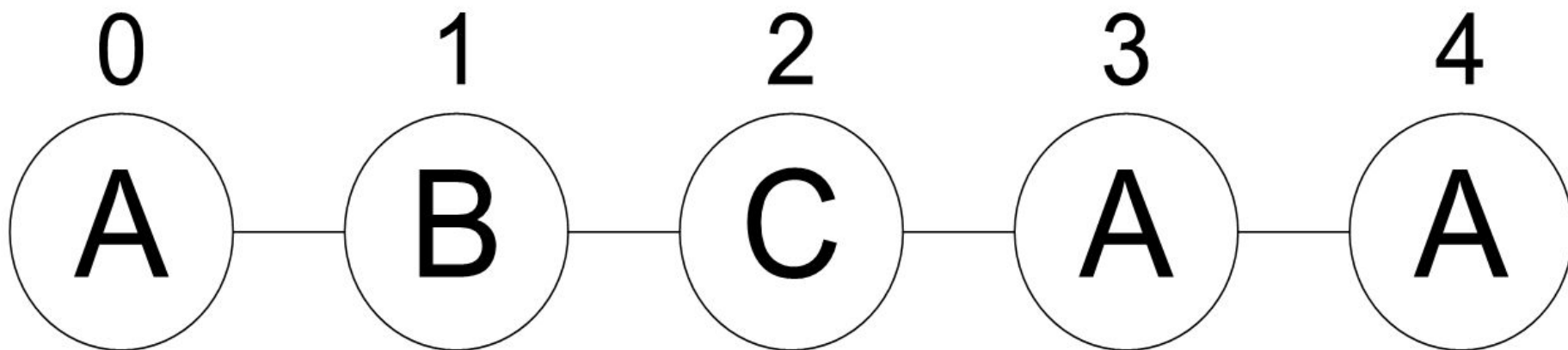
```
static void fill(int[], int value)
```

сравнение (9)

```
static boolean equals(int[] a1, int[] a2)
```

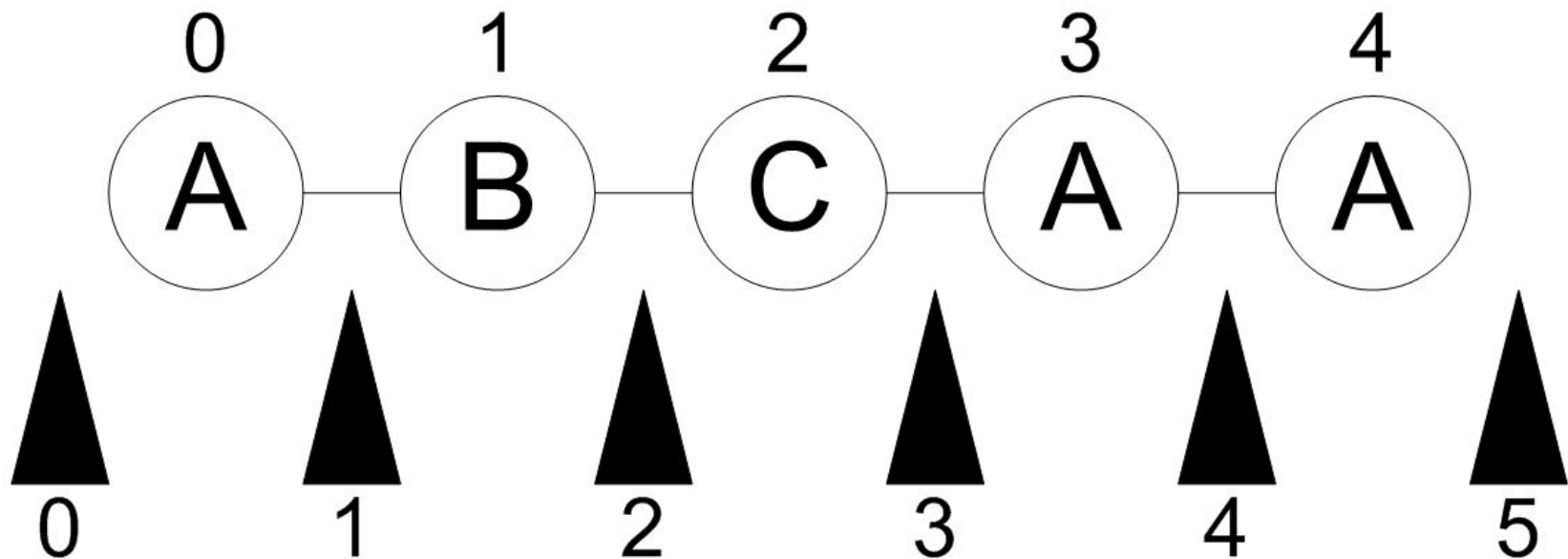
```
List<T> asList(T[] array), hashCode, toString()
```

- Список — коллекция с индексированными элементами
- Интерфейс **List**



- Доступ по индексу
 - `get(int i)` — чтение
 - `set(int l, Object e)` — запись
 - `add(int i, Object e)` — добавление
 - `remove(int i)` — удаление
- Поиск элементов
 - `indexOf(Object e)` — поиск с начала
 - `lastIndexOf(Object e)` — поиск с конца
- Взятие вида
 - `List subList(int from, int to)`

- Интерфейс `ListIterator` extends `Iterator`
- Метод `listIterator()`
- Предыдущий / Следующий элементы



- Передвижение

- `hasNext()` / `hasPrevious()` — проверка
- `next()` / `previous()` — взятие элемента
- `nextIndex()` / `previousIndex()` — определение индекса

- Изменение

- `remove()` — удаление элемента
- `set(Object e)` — изменение элемента
- `add(Object e)` — добавление элемента

- Все методы статические
- Константы EMPTY_LIST, EMPTY_SET, EMPTY_MAP
- Содержит методы для работы с коллекциями:
 - Поиска
 - Копирования списков
 - Определения частоты вхождения элемента
 - Позиции вхождения подсписка
 - Максимум, минимум
 - Обращения порядка
 - Перемешивания случайным образом
 - Сортировки
 - Синхронизации
 - Создания немодифицируемых видов

- Структура пакета java.lang:

http://www.inetworkgroup.net/0%27Reilly%20Reference%20Library/java/fclass/ch12_js.htm

- Работа с регулярными выражениями:

<http://www.javenue.info/post/43>

- Пакет java.util:

http://www.unix.com.ua/orelly/java-ent/jnut/ch23_01.htm

<http://www.intuit.ru/department/pl/javapl/14/2.html>

<http://www.realcoding.net/article/view/1981>

Thank you!

