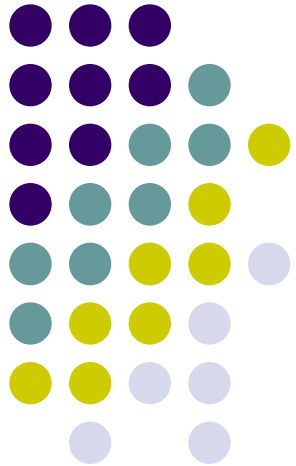
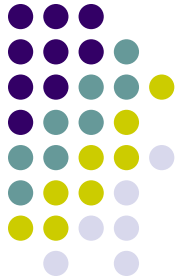
Коллекции в Java



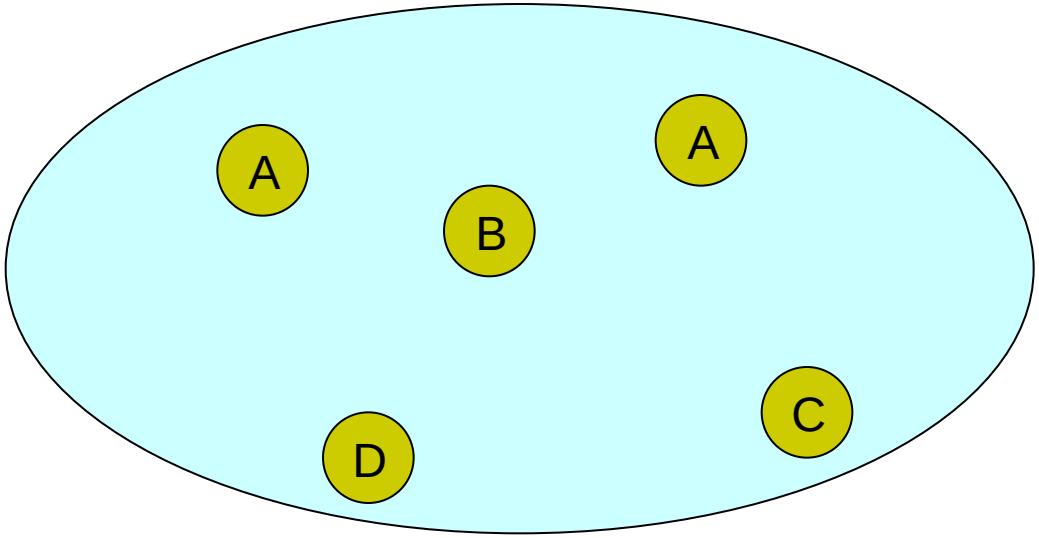
Java 1



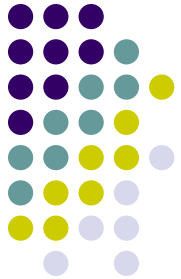
- Классы
  - Vector, Stack, Hashtable, BitSet
- Интерфейс
  - Enumeration

Коллекции

- Коллекция — неупорядоченный набор элементов
- Интерфейс Collection

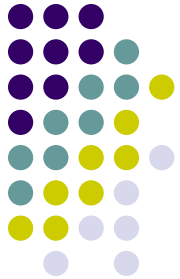


Интерфейс Collection
java.util.Collection<E>



`boolean add(E element)`	Добавляет элемент в набор данных. Возвращает `false`, если не может добавить аргумент.
`void clear( )`	Удаляет все элементы.
`boolean contains(Object obj)`	`true`, если набор данных содержит элемент `obj`.
`boolean isEmpty( )`	`true`, если набор данных не имеет элементов.
`Iterator<E> iterator( )`	Возвращает итератор, используемый для обращения к элементам.
`boolean remove(Object obj)`	Если аргумент присутствует в наборе данных, один экземпляр этого элемента будет удален. Возвращает `true`, если произошло удаление.
`int size( )`	Возвращает текущее количество элементов в наборе данных.
`Object[] toArray( )`	Возвращает массив, содержащий все элементы.

Интерфейс Collection<E>

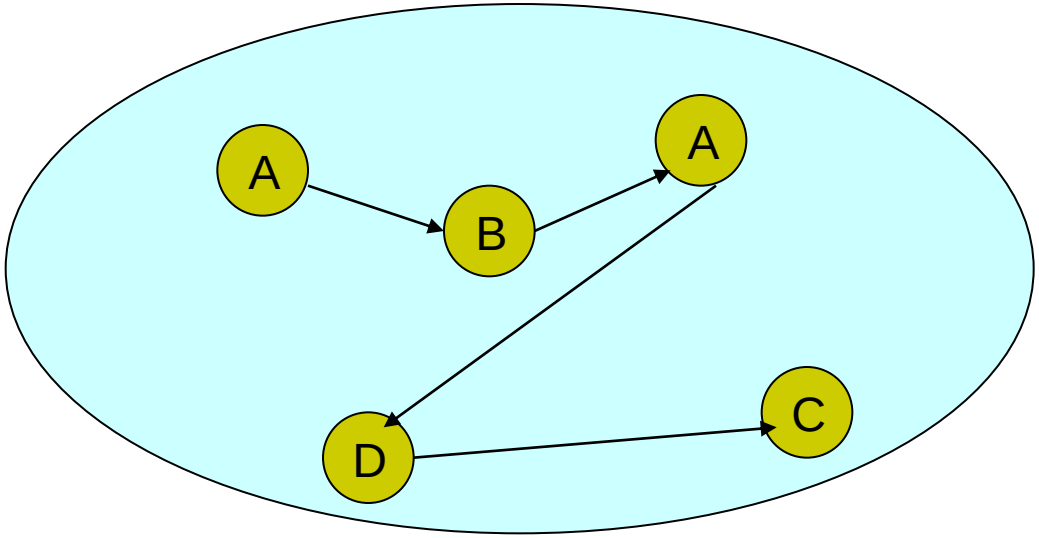


- Немодифицирующие операции
  - size()
  - isEmpty()
  - contains(Object o)
  - containsAll(Collection c)
- Модифицирующие операции
  - add(Object e)
  - addAll(Collection c)
  - remove(Object e)
  - removeAll(Collection c)
  - retainAll(Collection c)
  - clear()

✓ Исключения
- UnsupportedOperationException

Итераторы

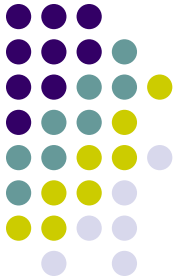
- Итератор — для обхода коллекции
- Интерфейс Iterator
- Метод Iterator Collection.iterator()



Интерфейс Iterator
java.util.Iterator<E>

boolean hasNext()	Возвращает `true`, если существует следующий элемент, к которому можно обратиться.
E next()	Возвращает следующий элемент. Генерируется исключение **`NoSuchElementException`**, если достигнут конец контейнера.
void remove()	Удаляет последний просмотренный элемент. Этот метод должен вызываться сразу после метода `next()`. Если этого не сделать генерируется исключение **`IllegalStateException.`** Если после чтения элемента набор данных изменился, данный метод генерирует исключение **`ConcurrentModificationException`**

Пример использования



```
import  java.util.*;
public class SimpleCollection {
  public static void main(String[ ] args) {
//Используем интерфейс, т.к. просто будем
// работать с особенностями Collection
Collection<String> c = new ArrayList<String>();
for(int i = 0; i < 10; i++)
    c.add(Integer.toString(i));
Iterator<String> it = c.iterator();
while(it.hasNext())
      System.out.println(it.next());
}
}
```

В JDK 5.0 цикл “for each”

```
for (String el : c){
 System.out.println(el);
}
```

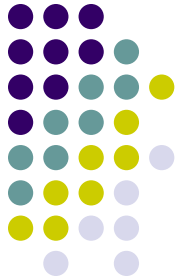
Компилятор преобразует такой цикл в цикл с итератором

Особенности использования метода remove()

```
Iterator<String> it = c.iterator();
it.next(); //пропускаем первый элемент
it.remove(); //удаляем его
```

При вызове метода next() итератор “перескакивает” через следующий элемент и возвращает ссылку на него

Класс AbstractCollection

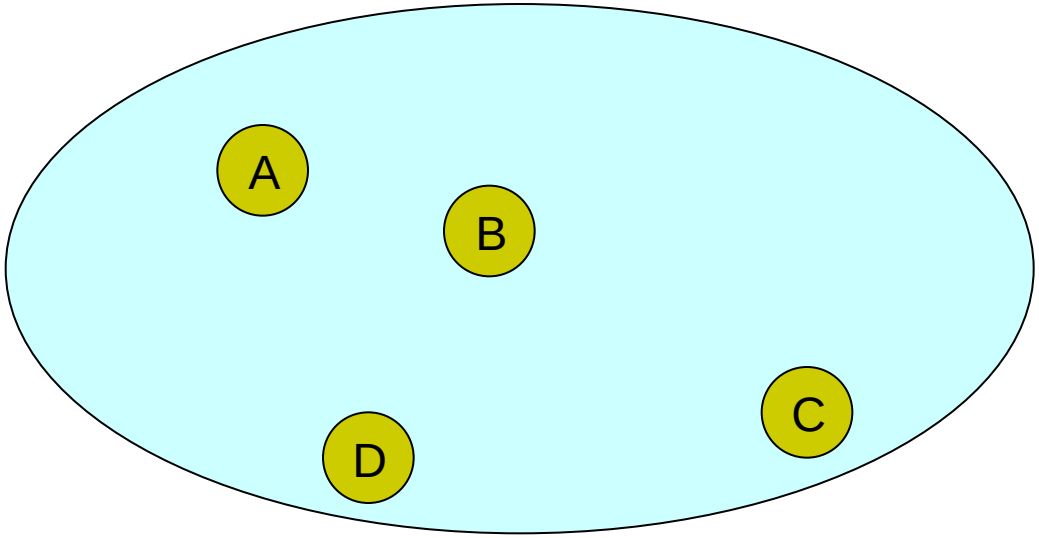


Позволяет быстро реализовывать коллекции

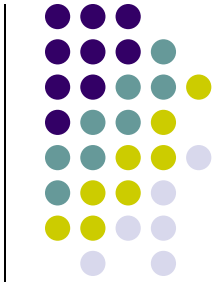
- Реализация неизменяемых коллекций
  - iterator()
  - size()
- Реализация изменяемых коллекций
  - add(Object o)
  - iterator.remove()

Множества

- Множество — коллекция без повторяющихся элементов
- Интерфейс Set



Сравнение элементов



Метод Object.equals(Object object)

- Рефлексивность o1.equals(o1)==true
- Симметричность o1.equals(o2) == o2.equals(o1)
- Транзитивность

o1.equals(o2) && o2.equals(o3) => o1.equals(o3)

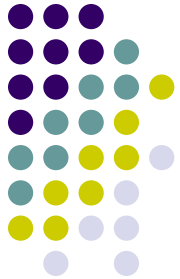
- Устойчивость

o1.equals(o2) не изменяется, если o1 и o2 не изменяются

- Обработка null

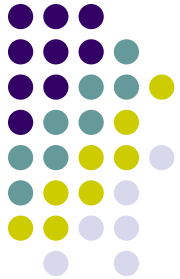
  o1.equals(null) == false

Интерпретация операций над множествами



- addAll(Collection c) – объединение множеств
- retainAll(Collection c) – пересечение множеств
- containsAll(Collection c) – проверка вхождения
- removeAll(Collection c) – разность множеств

Классы HashSet и TreeSet



- HashSet — множество на основе хэш-таблицы
- TreeSet — отсортированное множество (на основе дерева)

Вычисление хэш-кода

Метод Object.hashCode()

- Устойчивость

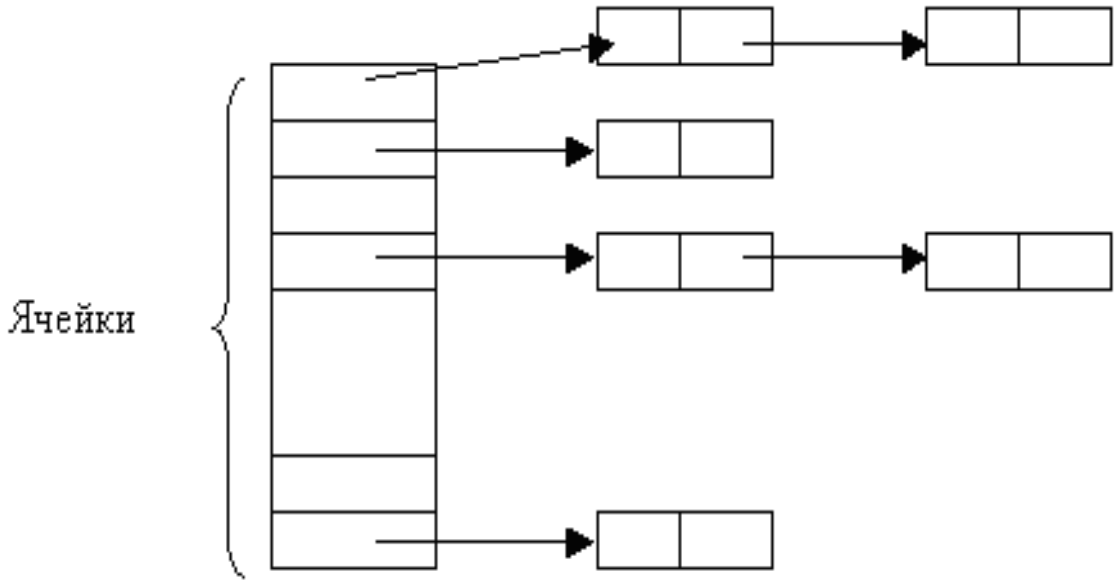
hashCode() не изменяется, если объект не изменяется

- Согласованность с equals()

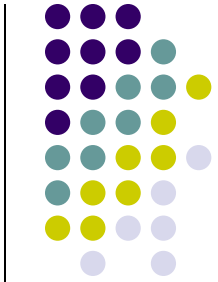
o1.equals(o2) =>

o1.hashCode() == o2.hashCode()

Хеш-таблица

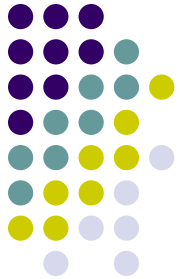


Пример использования



```
import java.util.*;
public class SetTest {
  public static void main(String [] args){
    Set<String> words = new HashSet<String>();
    Scanner in = new Scanner(System.in);
    while (in.hasNext()){
      String word = in.next();
      words.add(word);
    }
    System.out.println(”? ”+words.contains(”one”));
    Iterator <String> iter = words.iterator();
    while (iter.hasNext()){
       System.out.println(iter.next());
    }
  }
}
```

Упорядоченные множества



- Нужно, чтобы класс, которому принадлежат объекты, составляющие TreeSet, реализовывал
  - интерфейс Comparable
  - или интерфейс Comparator

Интерфейс Comparable

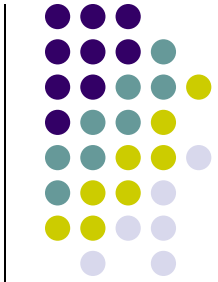
- int compareTo(Object o) — естественный порядок

0 – равны

«-» - меньше

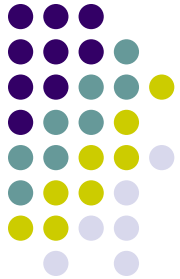
«+» - больше

Пример использования



```
class Item implements Comparable<Item> {
    public int compareTo(Item other){
        return intNumber – other.intNumber;
    }
. . . . . .
    private int intNumber;
    private MyCl1 val;
    private double  s;
}
//----------------------------------использование--------------------------------
TreeSet <Item> tree = new TreeSet<Item>();
Item t = new Item(. . .);   tree.add(t);
. . . . .
for (Item el : tree) System.out.println(el);
```

Интерфейс Comparator



int compare(Object o1, Object o2) – сравнение элементов

Класс для реализации интерфейса Comparator не содержит полей – это *функциональный класс*.

Пример

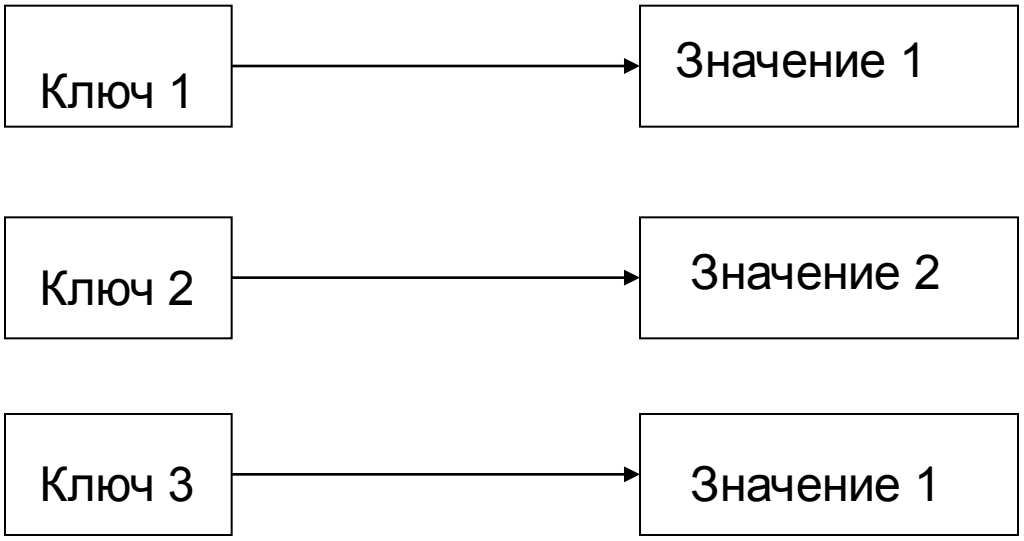
```
class ItemComp implements Comparator<Item>
public int compare (Item a,Item b) {
    MyCl1 v1 = a.getVal();
     MyCl1 v2 = b.getVal();
     return v1.compareTo(v2);
}
}

//----------------------использование---------------------
ItemComp comp = newItemComp();
TreeSet sortByVal = new TreeSet<Item>(comp);
```

Дополнительные операции

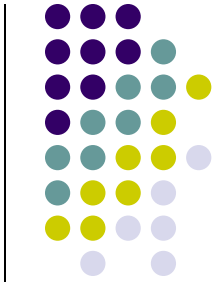
- first() – минимальный элемент
- last() – максимальный элемент
- headSet(Object o) – подмножество элементов меньших o
- tailSet(Object o) – подмножество элементов больших либо равных o
- subSet(Object o1, Object o2) – подмножество элементов меньших o2 и больше либо равных o1

Отображения (карты)

- Отображение - множество пар ключ-значение при уникальности ключа
- Интерфейс java.util.Map<K,V>

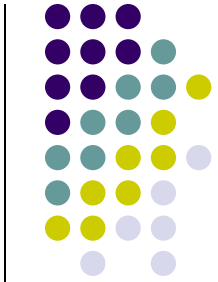


Методы



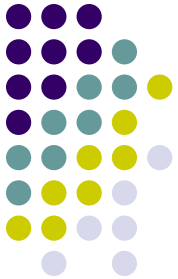
- Доступ
  - V get(K key) - получение значения по ключу или null
  - V put(K key, V value) – запись ключа и значения, возвращает старое значение по ключу или null
  - V remove(K key) – удаление значения по ключу, возвращает удаленное значение или null

Методы



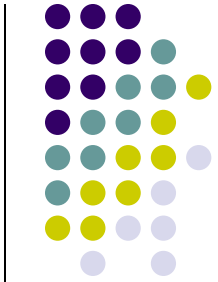
- Проверки
  - boolean containsKey(Object k) - наличие ключа
  - boolean containsValue(Object v) - наличие значения

Методы



- Дополнительные
  - void cleare() – очистка
  - int size() – размер
  - boolean isEmpty() – проверка на пустоту

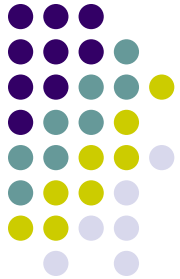
Методы



- Формирование представлений
  - Set <K> keySet() - возвращает представление карты в виде множества всех ключей.
  - Collection <V> values() -возвращает представление карты в виде множества всех значений
  - Set <Map.Entry<K,V>> entrySet() -возвращает представление карты в виде множества объектов Map.Entry, т.е. пар “ключ – значение”

Из каждого представления можно удалять элементы. При этом ключи и соответствующие им значения удалятся из карты. Добавлять новые элементы нельзя

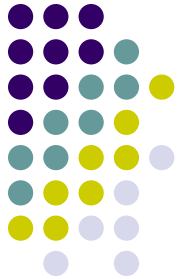
Пары



Внутренний класс java.util.Map.Entry <K,V>

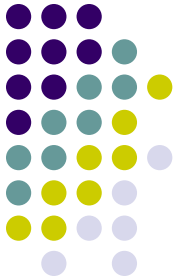
- Методы
  - K getKey()
  - V getValue()
  - V setValue(V value)

Реализации карт



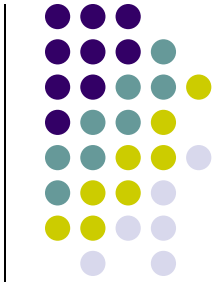
- Хеш-карта HashMap
- Карта-дерево TreeMap

Пример использования



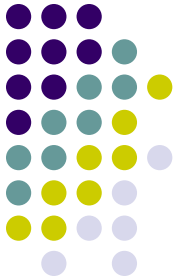
```
Map<String, Abonent> sprav=
new HashMap<String, Abonent>();

sprav.put(”+7(863)222-22-22”,
new Abonent(”FIO”,
                ”address”,”tariff”,...));

Abonent a=sprav.get(”+7(863)222-22-22”);
```

Пример использования



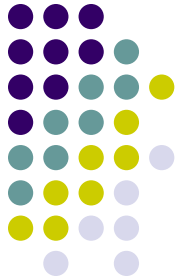
```
Set<String> tel=sprav.keySet();
for (String numb:tel)
{
  System.out.println(numb);
}
```

Пример использования



```
for (Map.Entry<String,Abonent> pair :
sprav.entrySet())
{
  String tel = pair.getKey();
  Abonent ab = pair.getValue();
. . .
}
```

Устаревшие коллекции



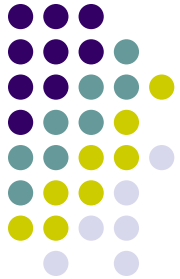
Устаревшие коллекции являются синхронизированными

- Vector
- Stack
- Dictionary
- Hashtable
- Enumeration

Устаревшие коллекции

Класс BitSet предназначен для работы с последовательностями битов. Каждый компонент этой коллекции может принимать булево значение, которое обозначает, установлен бит или нет. Содержимое BitSet может быть модифицировано содержимым другого BitSet с использованием операций AND, OR или XOR

Класс Properties

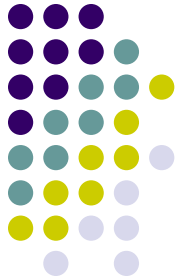


Класс Properties - предназначен для хранения множества свойств (параметров).

Это карты особого вида:

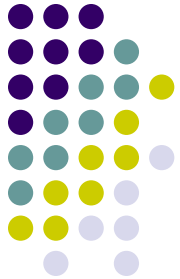
- ключ и значение являются строками
- карту можно сохранить в файле и загрузить из файла
- для используемых по умолчанию значений создается вторая карта

Методы



- String getProperty(String key)
- String getProperty(String key, String defaultValue)
- void setProperty(String key, String value)
- void load(InputStream in)
- void store(OutputStream out, String header)

Алгоритмы



Класс Collections

- Алгоритмы для работы с коллекциями
  - Простые операции
  - Перемешивание
  - Сортировка
  - Двоичный поиск
  - Поиск минимума и максимума
- Специальные коллекции
- Оболочки коллекций

Простые операции

- Заполнение списка указанным значением

```
void fill(List <? super T> l, T v)
```

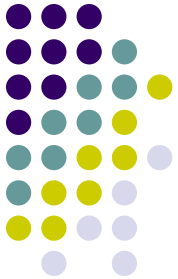
- Переворачивание списка

```
void reverse(List <?> l)
```

- Копирование из списка в список

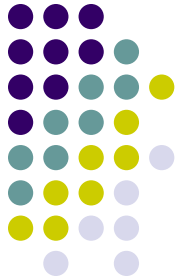
```
void copy(List <? super T> to, List <T> from)
```

Перемешивание и сортировка



- void shuffle(List <?> e)
- void shuffle(List <?> e, Random r)
- void sort (List<T> e)
- void sort (List <T> e,
  Comparator <?super T> c)

Двоичный поиск



- int binarySearch(List<T> e, T key)
- int binarySearch(List<T> e, T key,
  Comparator <? super T> c)