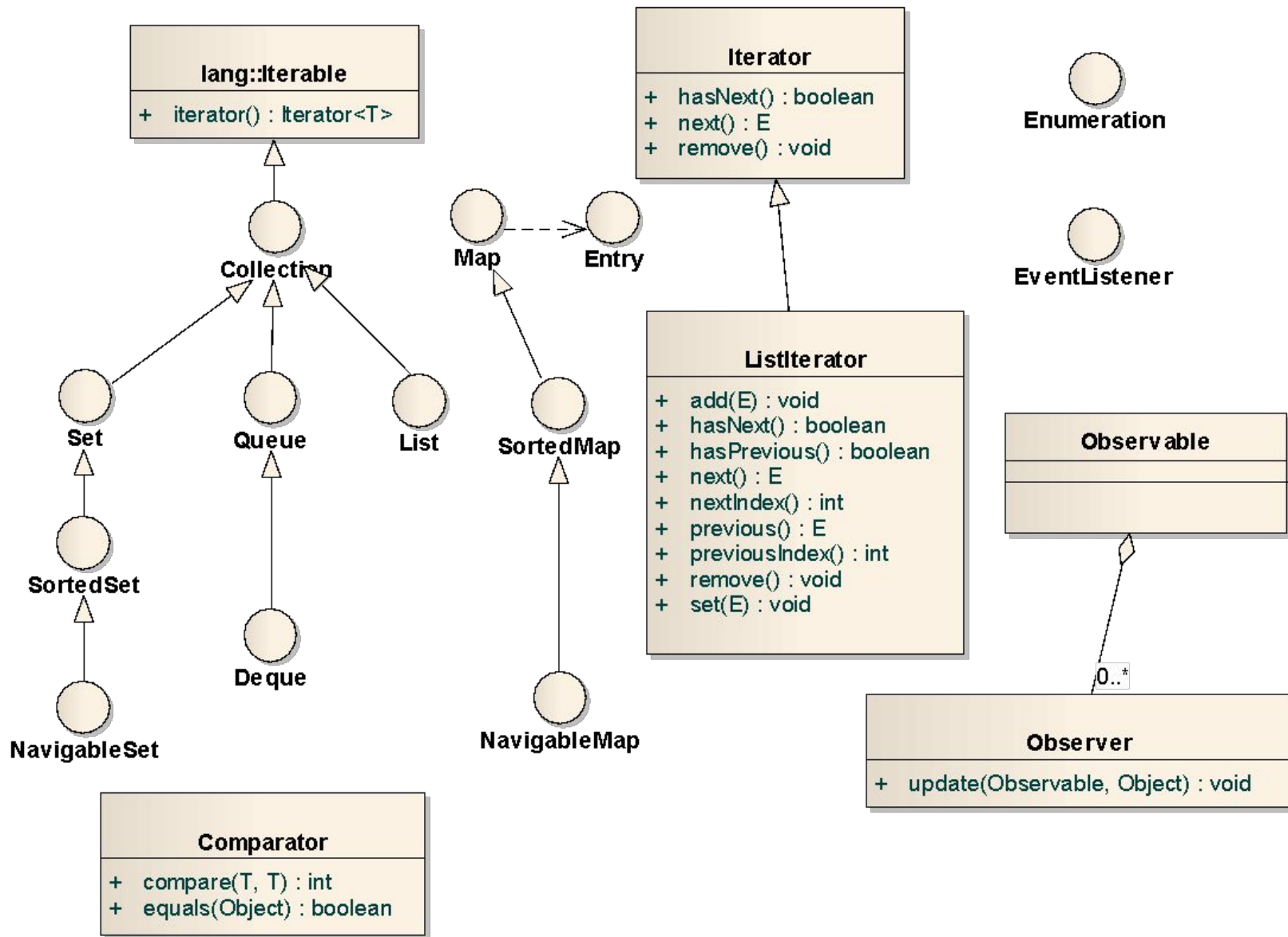

Основы программирования на языке Java

Стандартная библиотека Java:
java.util, java.io, java.net

class Overview



Enumeration, Comparator

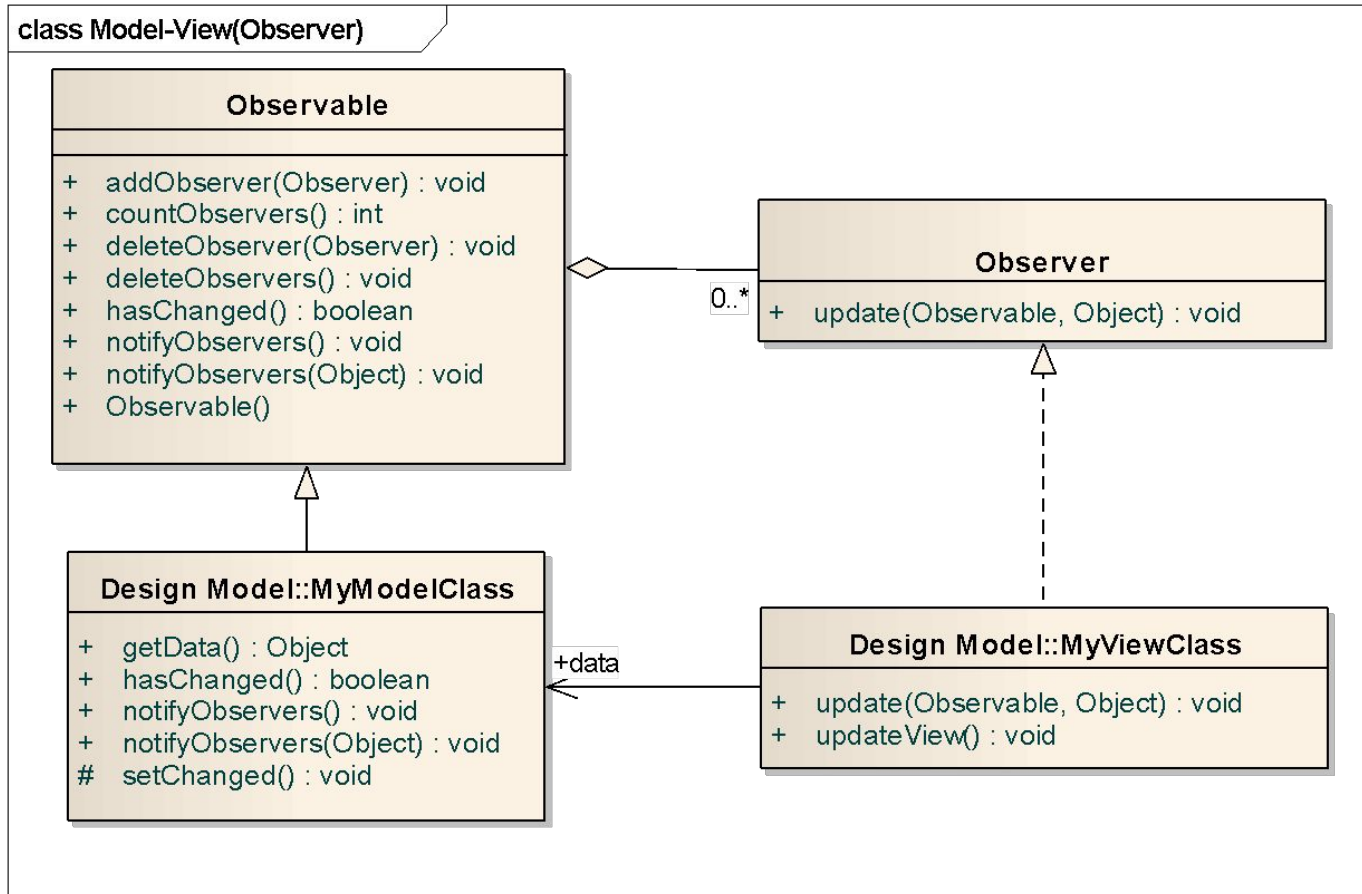
- Enumeration – объекты классов, реализующих данный интерфейс, используются для предоставления однопроходного последовательного доступа к серии объектов:

```
Hashtable<String, String> t = ...;  
for(Enumeration<String> e = t.keys(); e.hasMoreElements();) {  
    String s = e.nextElement();  
}
```

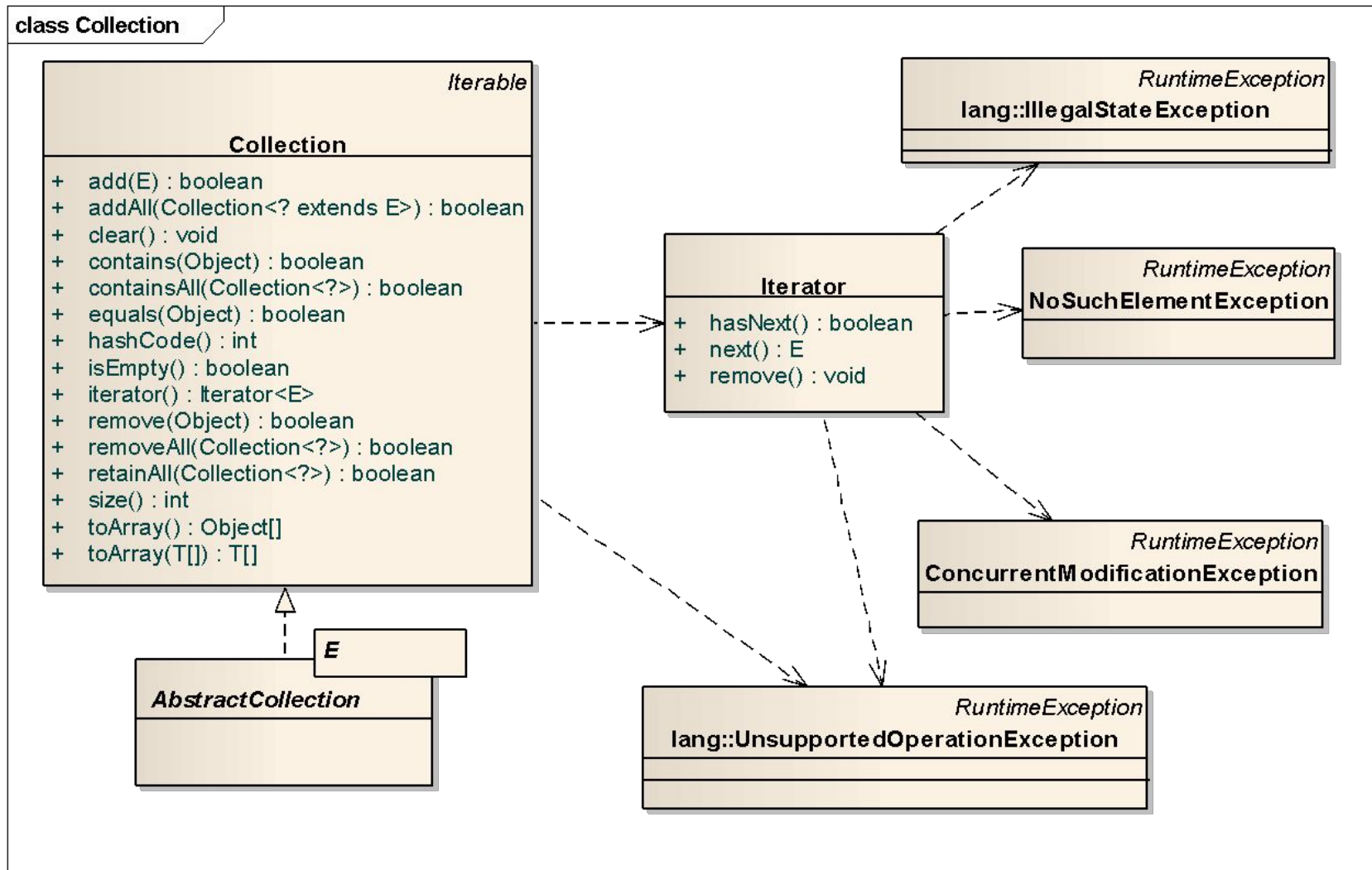
- Comparator – классы, реализующие данный интерфейс используются для обеспечения сравнения произвольных элементов. Используются при сортировке и организации упорядоченных контейнеров:

```
Comparator<MyClass> c = new Comparator<MyClass> () {...};  
MyClass array[] = ...;  
Arrays.sort(array, c);
```

Паттерн Model-View (Observer)



Коллекция объектов (контейнер)

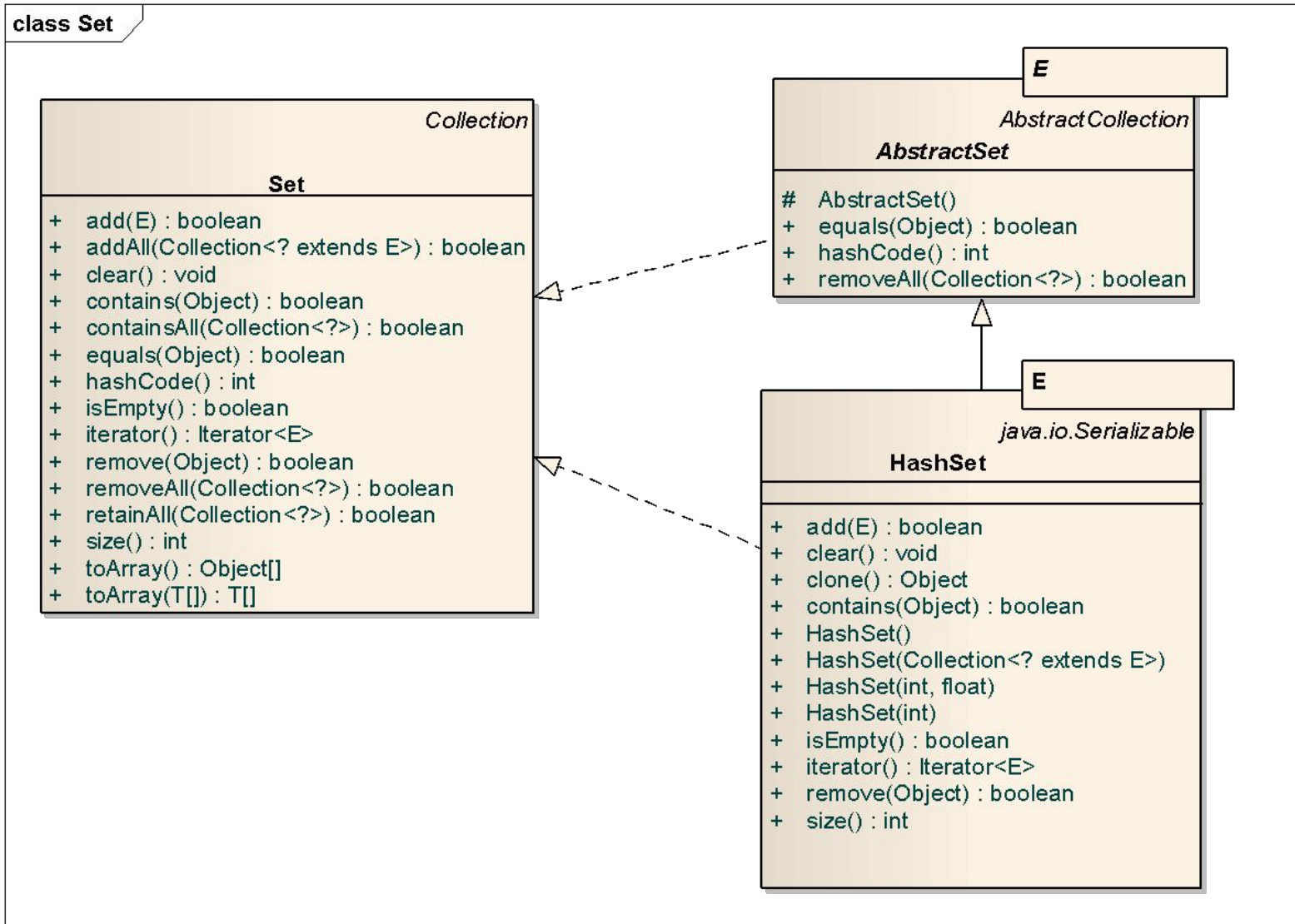


Использование итераторов

```
Collection<String> col = ...;  
for (Iterator<String> i = col.iterator(); i.hasNext(); ) {  
    String s = i.next();  
}  
for (String s : col ) { s; } //Альтернатива
```

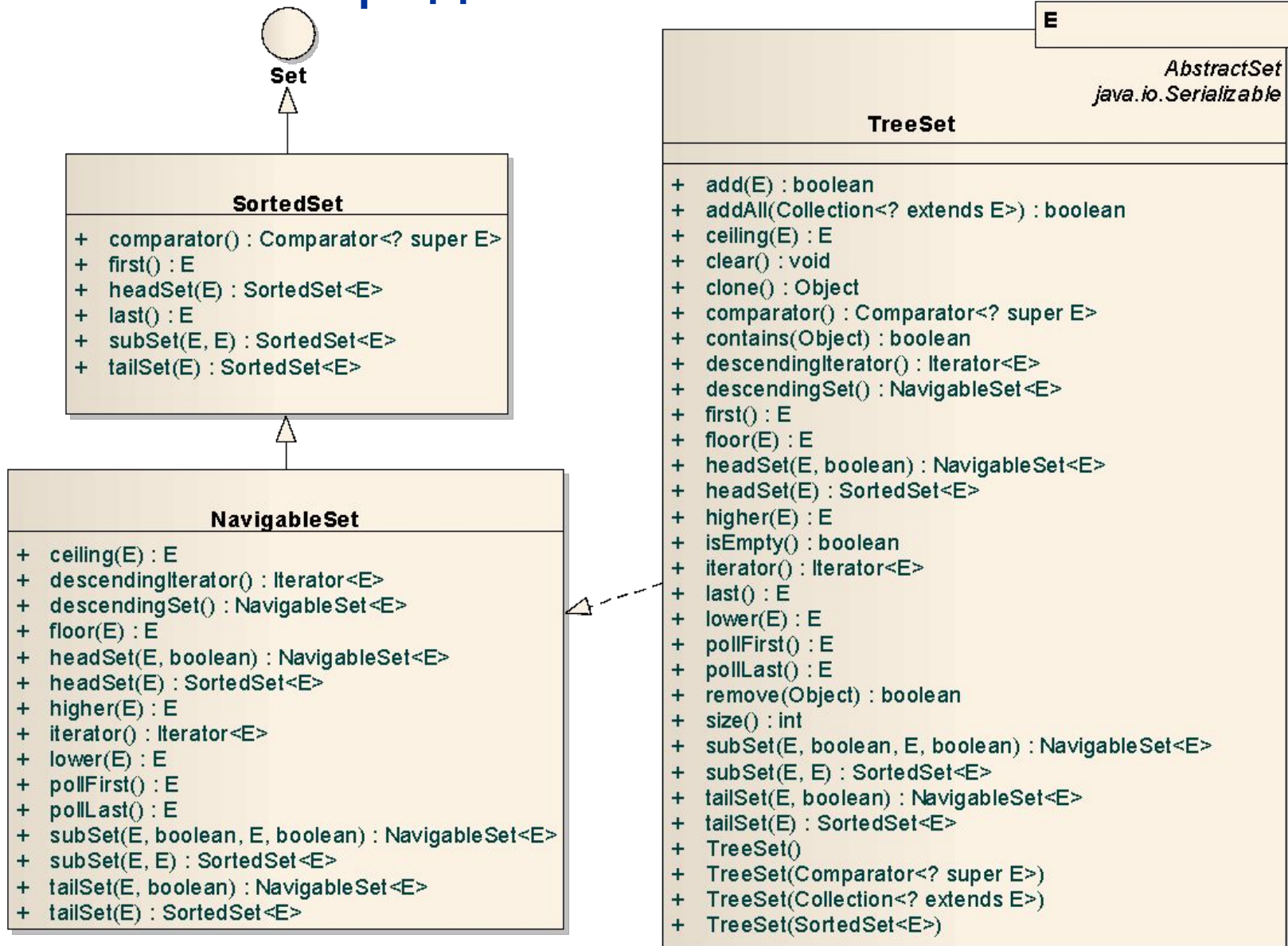
- ❑ **ConcurrentModificationException** – возникает когда коллекция изменена во время итерирования другим потоком
- ❑ **IllegalStateException** – возникает при попытке удаления элемента с помощью вызова `remove()`, когда еще не был вызван первый метод `next()`, или итератор указывает на конец коллекции (`hasNext()` возвращает `false`)
- ❑ **NoSuchElementException** – возникает при попытке вызова `next()` когда `hasNext()` возвращает `false`
- ❑ **UnsupportedOperationException** – возникает при вызове метода который не поддерживается конкретной реализацией коллекции

Множества

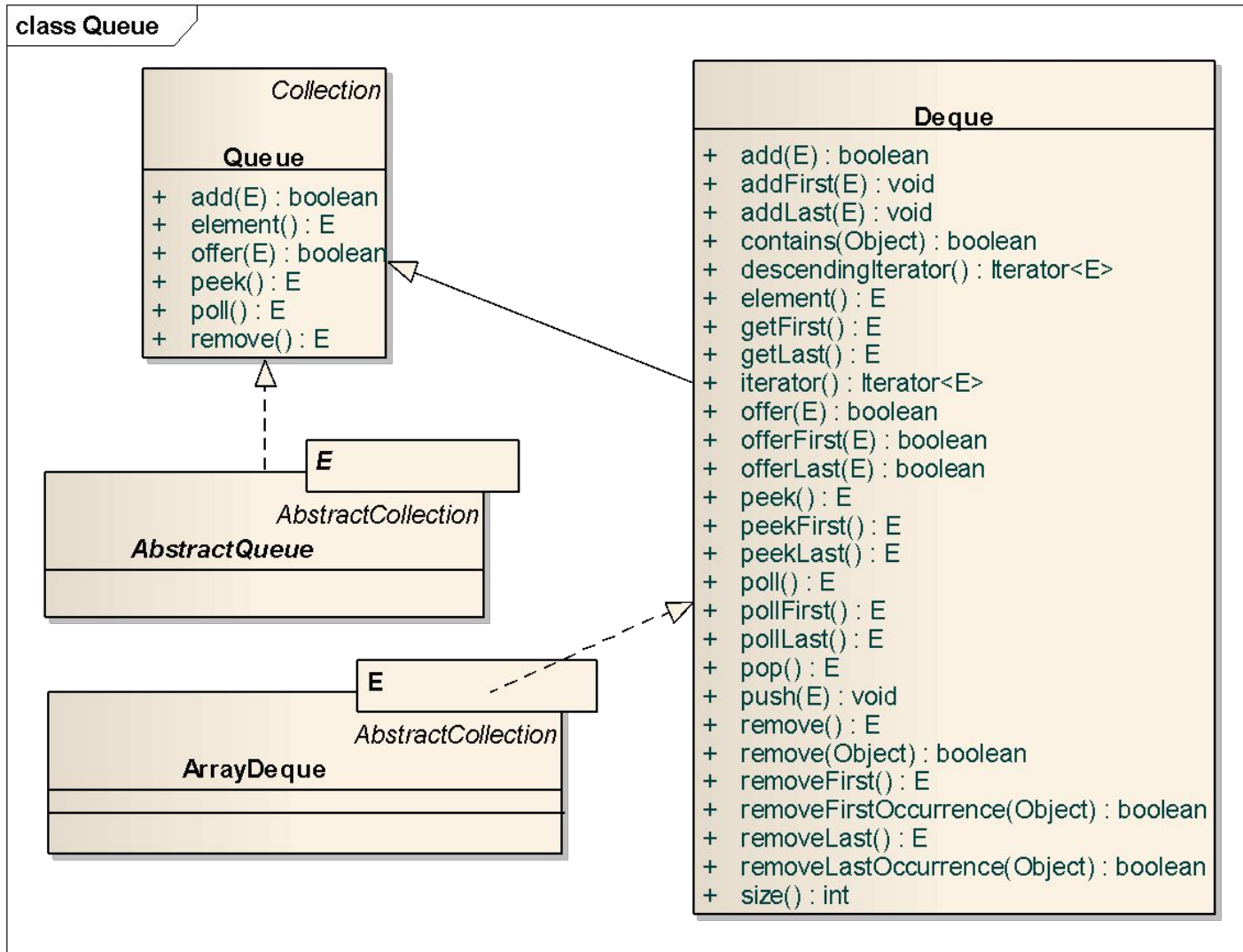


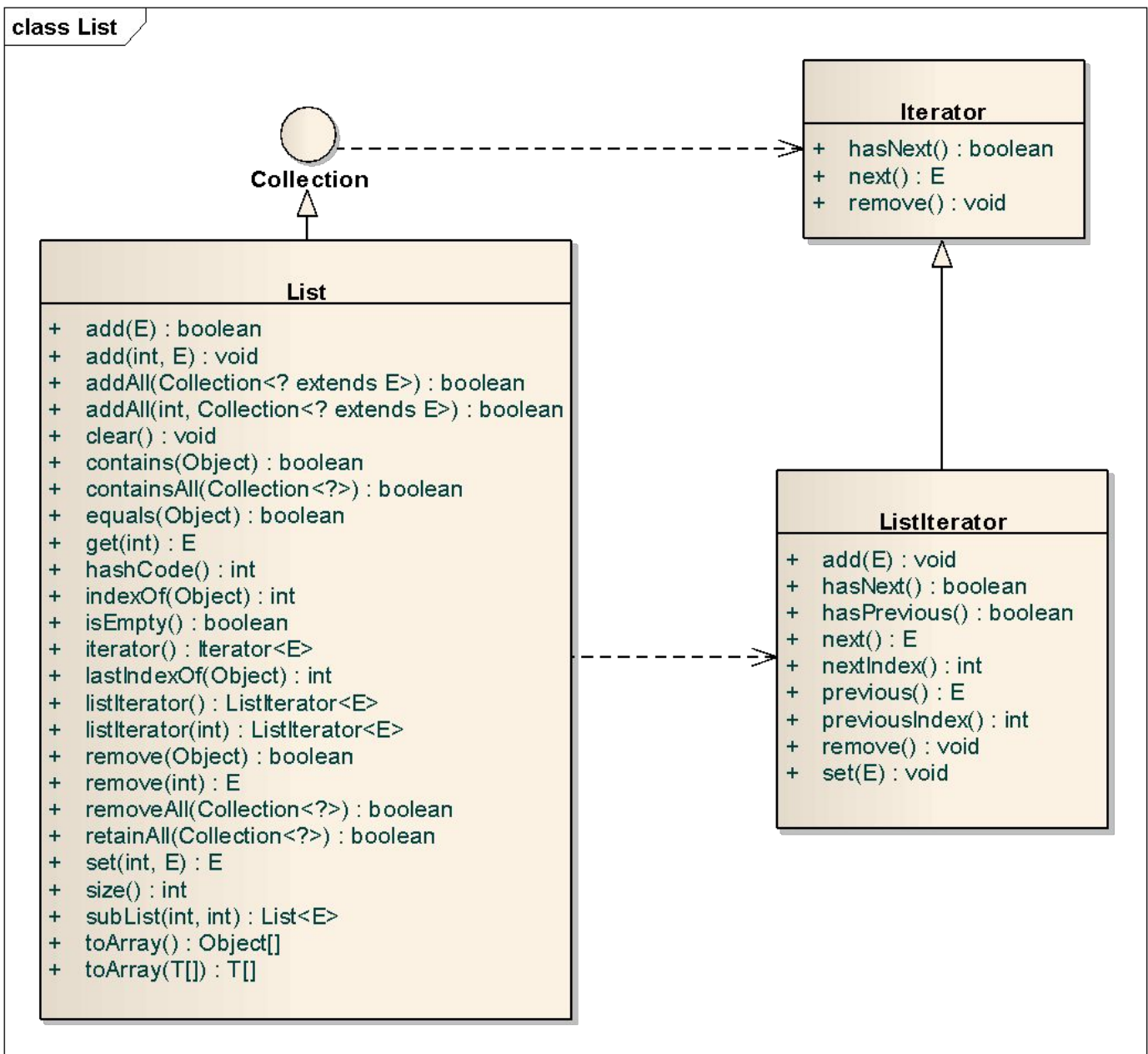
Упорядоченные множества

class SortedSet

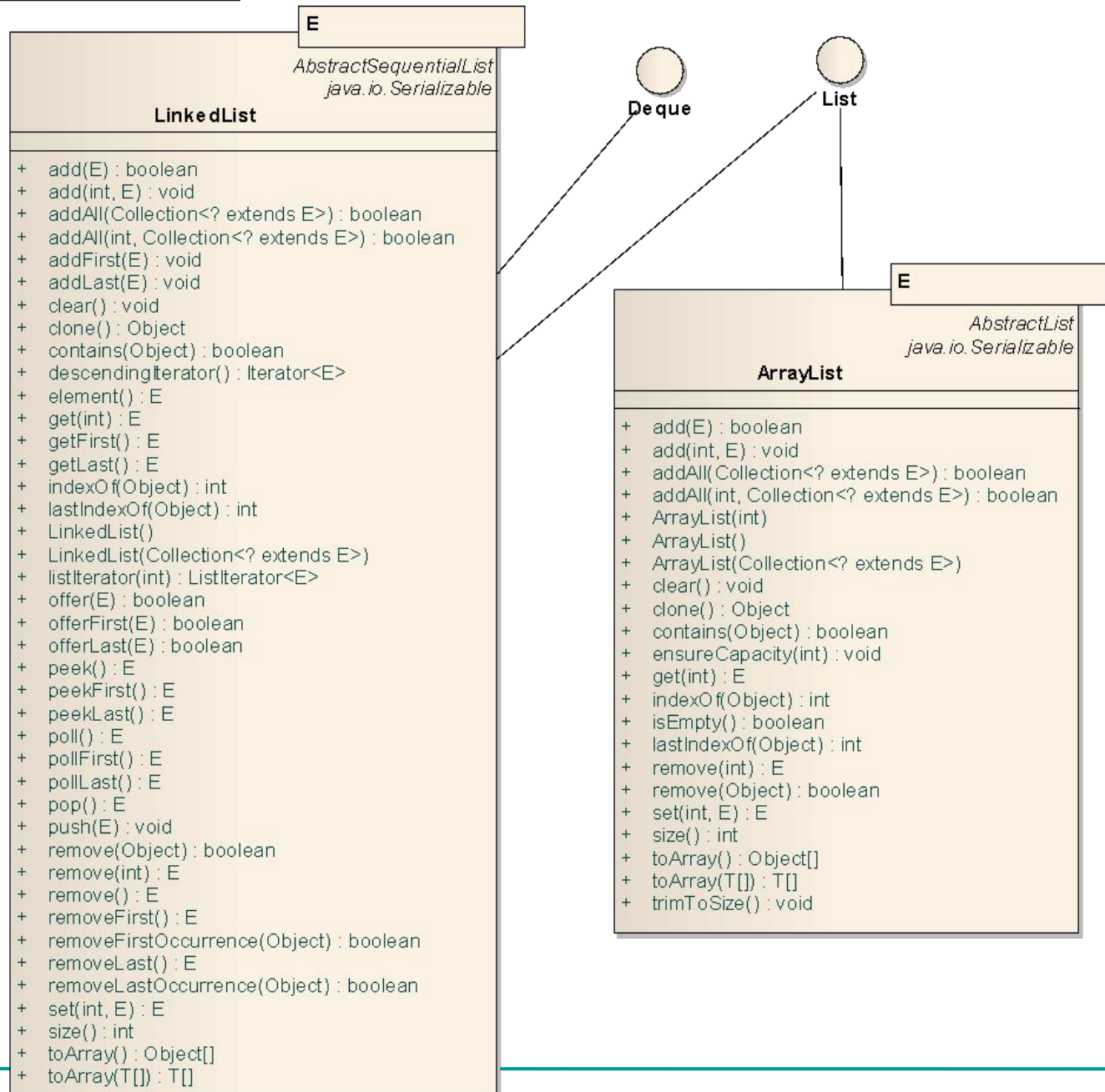


Очереди





Реализации списка

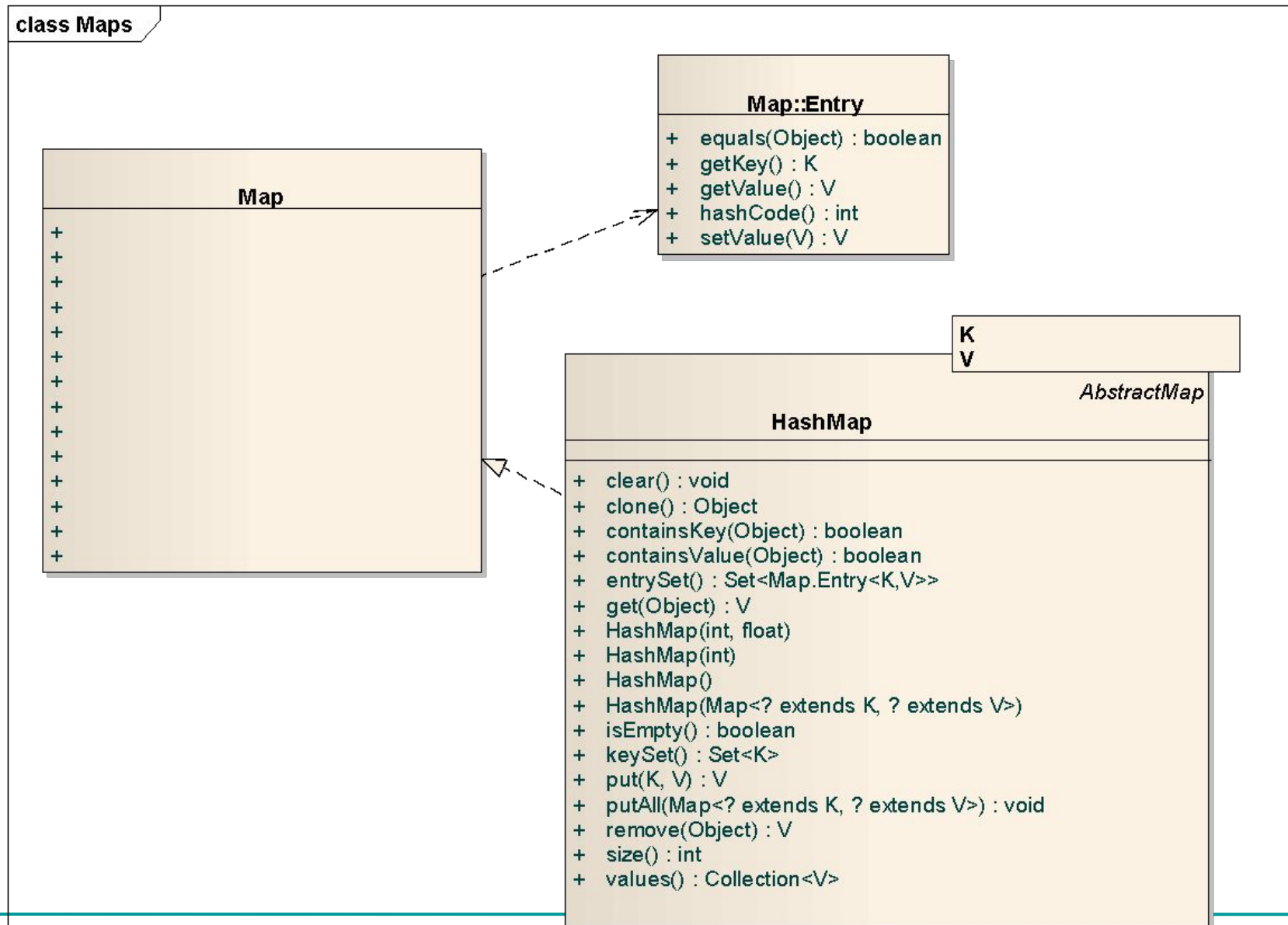


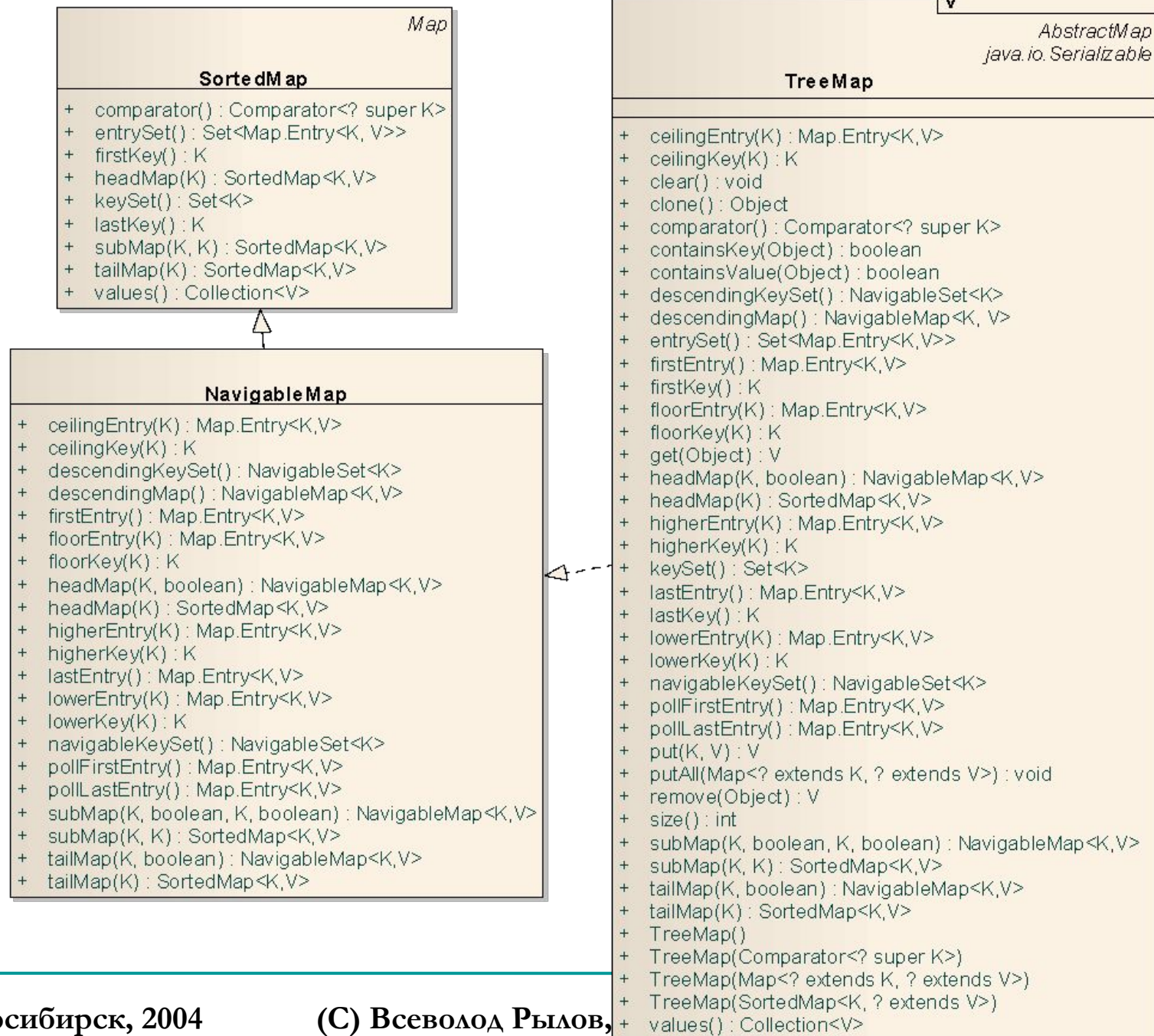
Пример простой реализации стека

```
import java.util.*;
public class Stack<T> {
    private LinkedList<T> content;
    public Stack() {
        content = new LinkedList<T>();
    }
    public void push(T obj) {
        content.addFirst(obj);
    }
    public T pop() throws
        NoSuchElementException
    {
        return content.removeFirst();
    }
    public boolean hasMoreElements() {
        if (content.size() > 0) return true;
        else return false;
    }
}
```

```
public static void main(String args[])
    throws NoSuchElementException
{
    Stack<String> s = new
Stack<String>();
    for (String o : args) {
        s.push(o);
    }
    while(s.hasMoreElements()) {
        System.out.println(s.pop());
    }
}
```

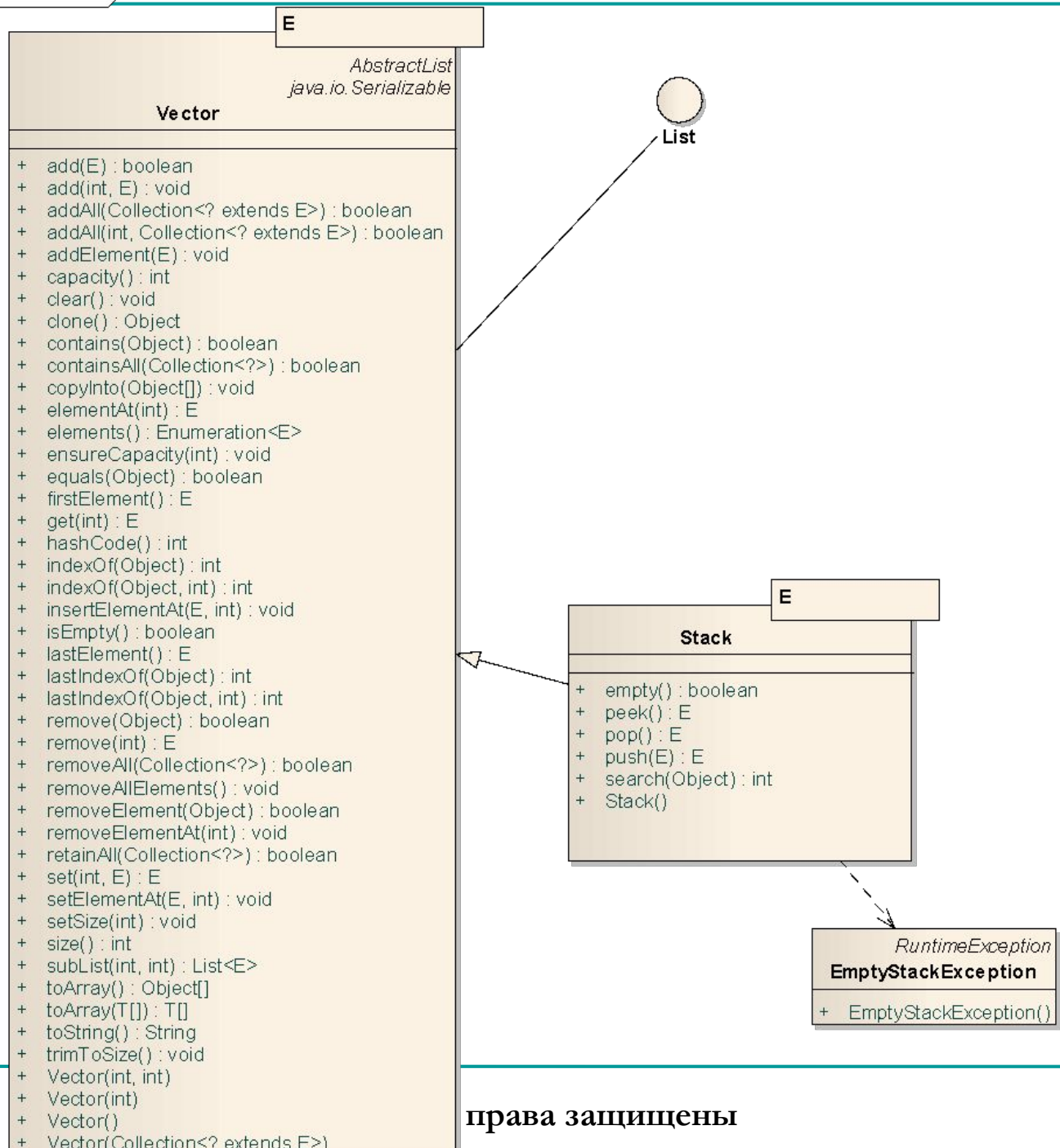
Ассоциативные контейнеры



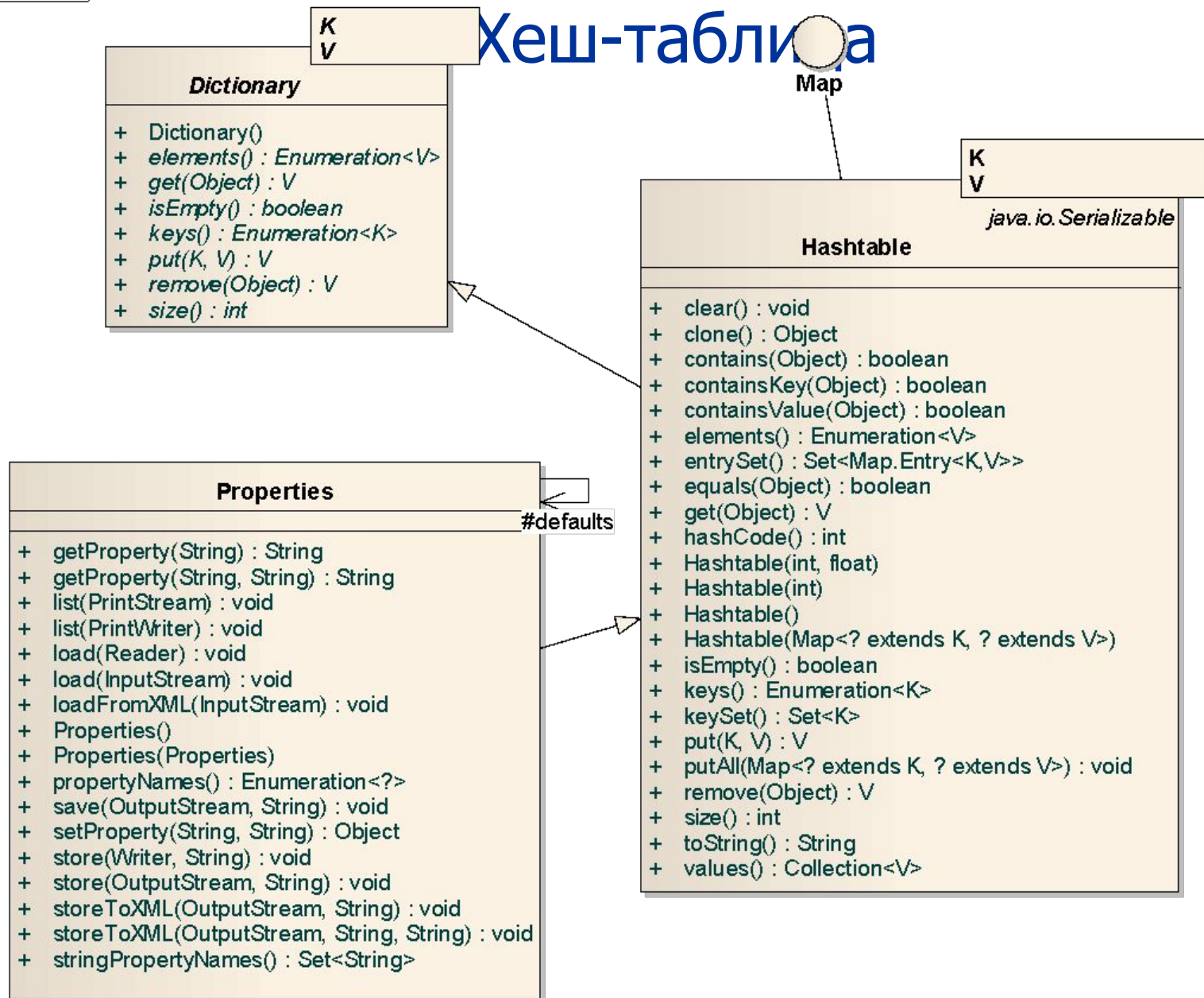


Вектор

class Vector



Хеш-таблица



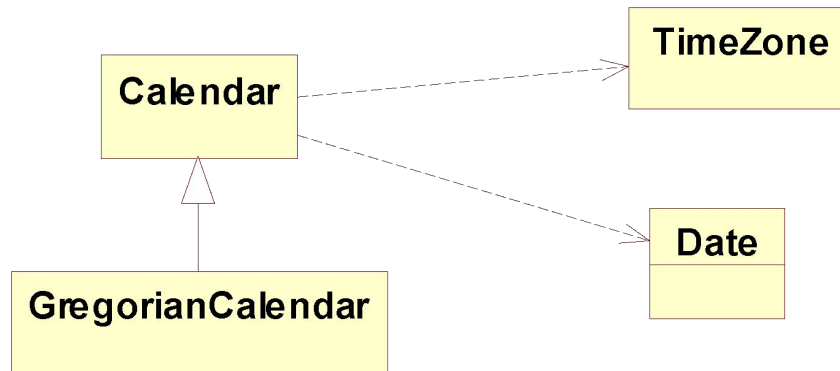
Класс-утилита Collections

- Служит для поддержки классов контейнеров и осуществления основных манипуляций с содержимым:
 - `sort(List)`, `sort(List, Comparator)`
 - `binarySearch(List, Object)`, `binarySearch(List, Object, Comparator)`
 - `reverse(List)`
 - `shuffle(List)`, `shuffle(List, Random)`
 - `fill(List, Object)`
 - `copy(List, List)`
 - `min(Collection)`, `min(Collection, Comparator)`
 - `max(Collection)`, `max(Collection, Comoarator)`
- Представляет коллекцию как Enumeration (метод `enumeration(Collection)`)
- Создает не модифицируемый список, содержащий несколько копий объекта: `nCopies(int, Object): List`
- Содержит специальные статические поля: `EMPTY_SET`, `EMPTY_LIST`, `EMPTY_MAP`

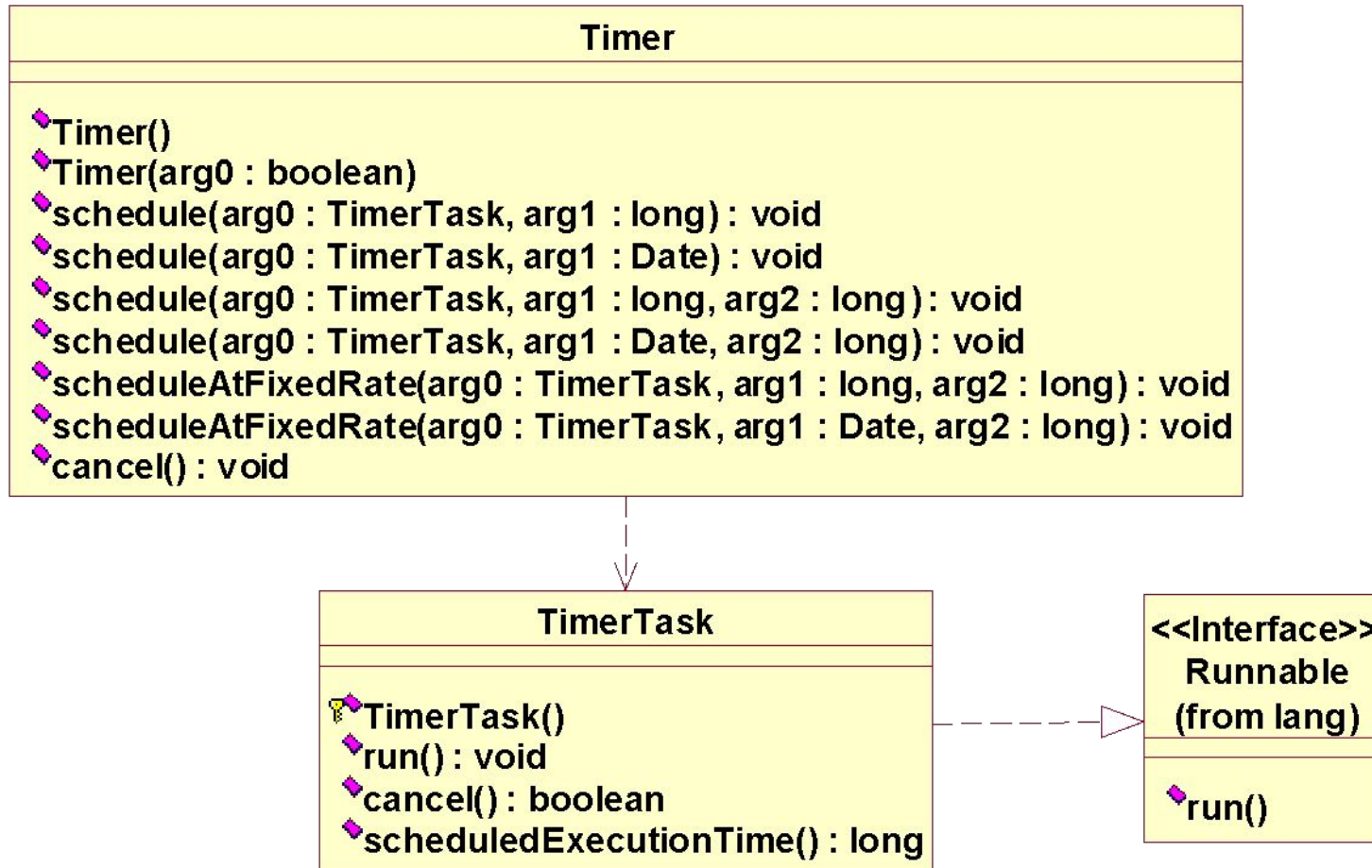
Класс-утилита Collections (продолж.)

- Предоставляет обертки для защиты контейнеров от модификации или для обеспечения синхронизированного доступа:
 - `unmodifiableXXX(arg:XXX):XXX`
 - `synchronizedXXX(arg:XXX):XXX`
 - `XXX` - это `Collection`, `Set`, `SortedSet`, `List`, `Map`, `SortedMap`
- Предоставляет не модифицируемые обертки-одиночки, содержащие только один элемент (или пару ключ-значение):
 - `singleton(Object):Set`
 - `singletonList(Object):List`
 - `singletonMap(Object, Object):Map`

Работа со временем



Таймер и планируемые задания

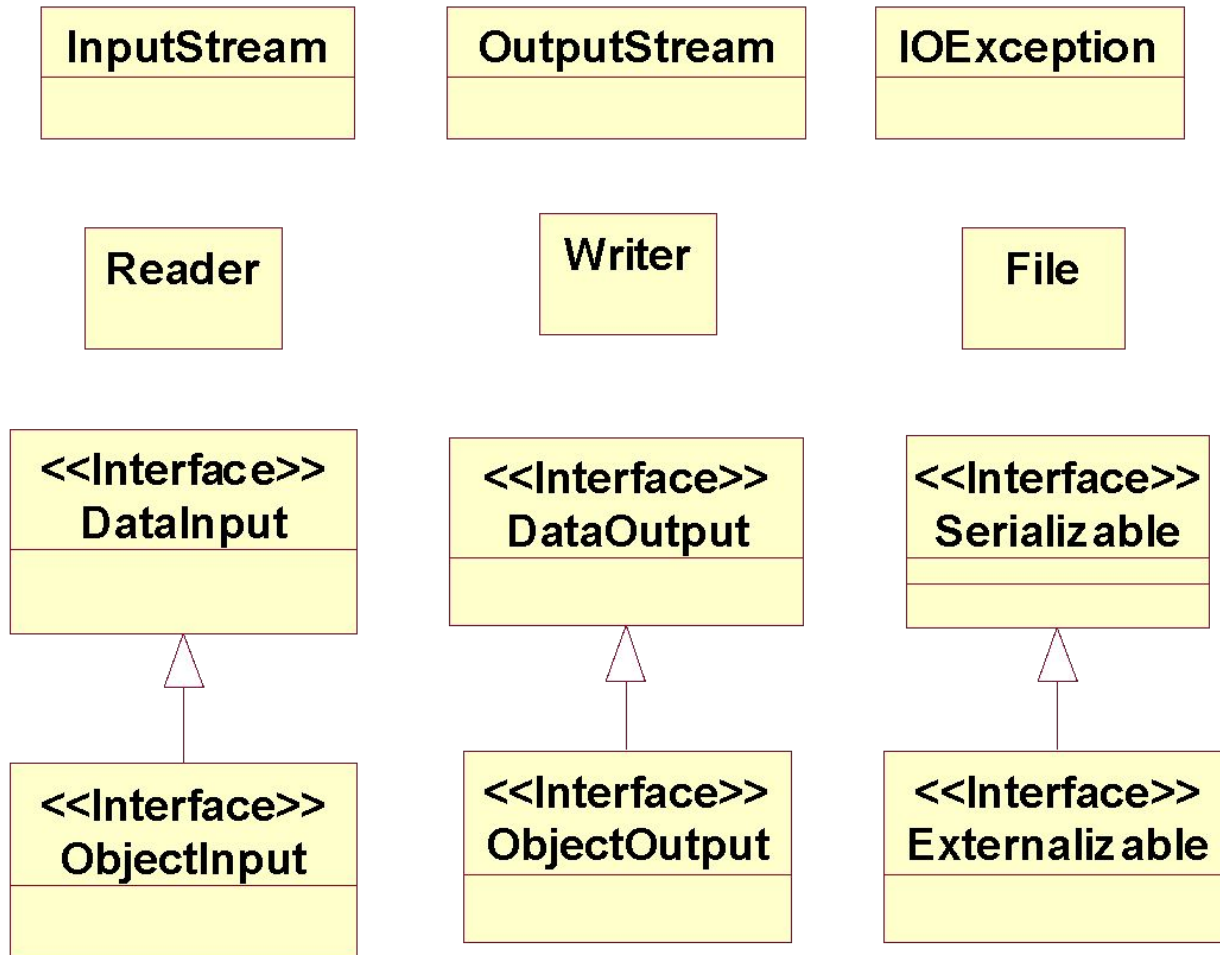


Интернационализация

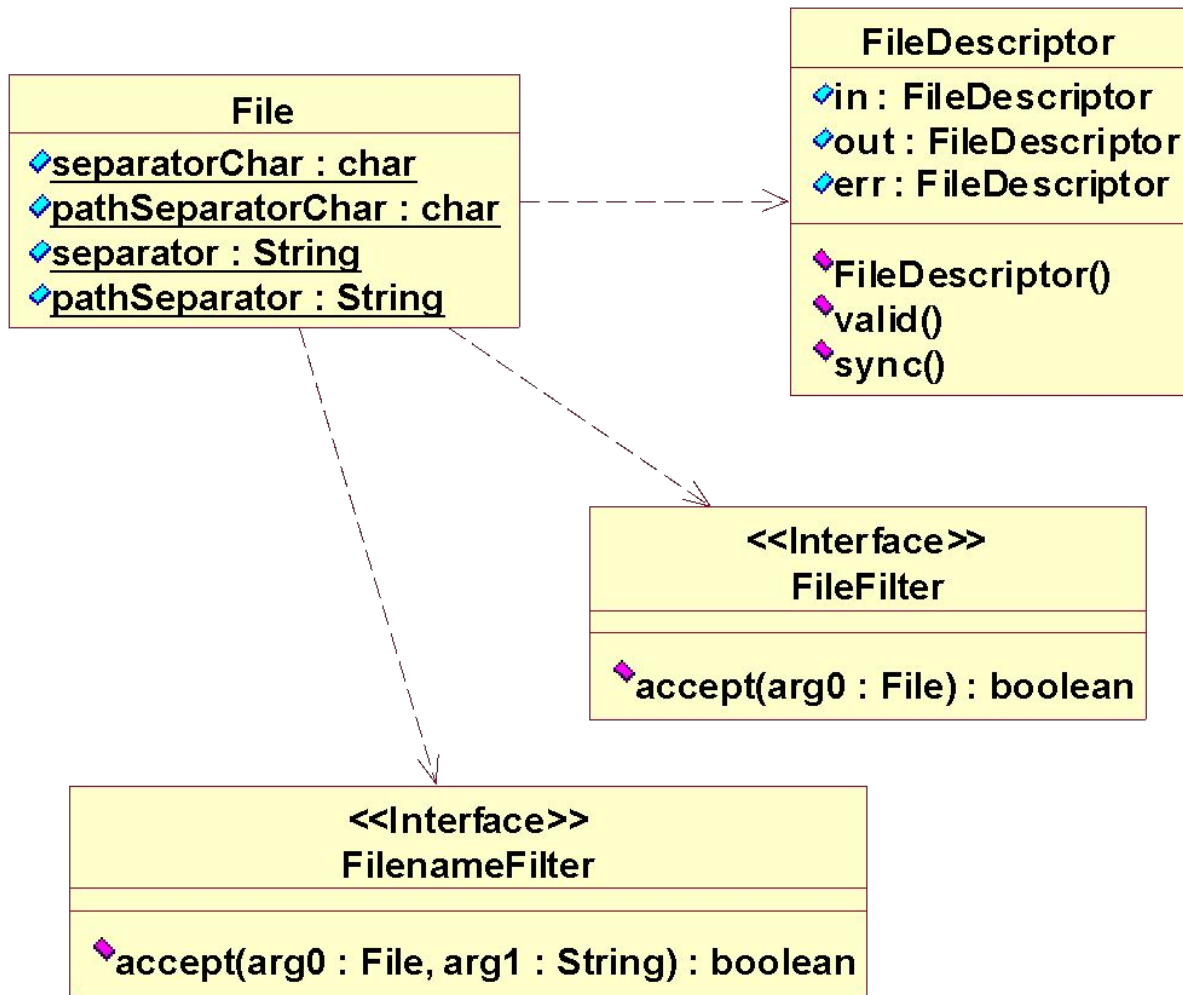
Locale

- ◆ `Locale(arg0 : String, arg1 : String, arg2 : String)`
- ◆ `Locale(arg0 : String, arg1 : String)`
- ◆ `getDefault() : Locale`
- ◆ `setDefault(arg0 : Locale) : void`
- ◆ `getAvailableLocales() : Locale[]`
- ◆ `getISOCountries() : String[]`
- ◆ `getISOLanguages() : String[]`
- ◆ `getLanguage() : String`
- ◆ `getCountry() : String`
- ◆ `getVariant() : String`
- ◆ `toString() : String`
- ◆ `getISO3Language() : String`
- ◆ `getISO3Country() : String`
- ◆ `getDisplayLanguage() : String`
- ◆ `getDisplayLanguage(arg0 : Locale) : String`
- ◆ `getDisplayCountry() : String`
- ◆ `getDisplayCountry(arg0 : Locale) : String`
- ◆ `getDisplayVariant() : String`
- ◆ `getDisplayVariant(arg0 : Locale) : String`
- ◆ `getDisplayName() : String`
- ◆ `getDisplayName(arg0 : Locale) : String`
- ◆ `clone() : Object`
- ◆ `hashCode() : int`
- ◆ `equals(arg0 : Object) : boolean`

Подсистема ввода-вывода java.io



Средства работы с файлами



Потоки ввода-вывода

- Потоки ввода вывода бывают двух типов:
 - узловые потоки (node streams) – предоставляют ввод вывод на уровне потоков байт
 - фильтрующие потоки (filter streams) – предоставляют обертки вокруг низкоуровневых потоков для обеспечения расширенной функциональности

InputStream

- ◆ `InputStream()`
- ◆ `read() : int`
- ◆ `read(arg0 : byte[]) : int`
- ◆ `read(arg0 : byte[], arg1 : int, arg2 : int) : int`
- ◆ `skip(arg0 : long) : long`
- ◆ `available() : int`
- ◆ `close() : void`
- ◆ `mark(arg0 : int) : void`
- ◆ `reset() : void`
- ◆ `markSupported() : boolean`

OutputStream

- ◆ `OutputStream()`
- ◆ `write(arg0 : int) : void`
- ◆ `write(arg0 : byte[]) : void`
- ◆ `write(arg0 : byte[], arg1 : int, arg2 : int) : void`
- ◆ `flush() : void`
- ◆ `close() : void`

Узловые потоки

- ❑ Узловые потоки работают непосредственно с подлежащими источниками (приемниками) данных на уровне байт.
- ❑ Источниками(приемниками) могут быть:
 - ❑ файлы на файловой системе – `FileInputStream`, `FileOutputStream`
 - ❑ массивы байт в памяти – `ByteArrayInputStream`, `ByteArrayOutputStream`
 - ❑ каналы (pipes) - `PipedInputStream`, `PipedOutputStream`
 - ❑ строки – `StringBufferInputStream` (deprecated)
 - ❑ прочие источники – сокеты и т.д: `something.getInputStream()`, `something.getOutputStream()`

Пример – копирование файлов

```
import java.io.*;
public class FileCopy {
    public static void main(String args[]) {
        InputStream in = null;
        OutputStream out = null;
        try {
            in = new FileInputStream(args[0]);
            out = new FileOutputStream(args[1]);
        } catch (Exception e) {
            System.out.println("Неправильные аргументы");
            if (in!=null) try {in.close();} catch(IOException ix2){}
            return;
        }
        byte buf[] = new byte[1024]; //Буфер для хранения промежуточного блока
        int read; //Переменная для хранения количества прочитанных байт
        try {
            while ( (read = in.read(buf)) >= 0) //При достижении конца вернет <0
                out.write(buf, 0, read); //Записываем ровно столько сколько удалось прочитать
        } catch (IOException ioex) {
            System.out.println("Ошибка ввода/вывода:"+ioex.getLocalizedMessage());
        } finally {
            try {in.close();} catch(IOException ix2){}
            try {out.close();} catch (IOException ix3) {}
        }
    }
}
```

Фильтрующие потоки











- ❑ Фильтрующие потоки предоставляют дополнительную функциональность на основе других потоков
 - ❑ абстрактные – `FilterInputStream`, `FilterOutputStream`
 - ❑ буферизация – `BufferedInputStream`, `BufferedOutputStream`
 - ❑ форматированный ввод-вывод данных – `DataInputStream`, `DataOutputStream`
 - ❑ сериализация объектов – `ObjectInputStream`, `ObjectOutputStream`
 - ❑ опережающее чтение – `PushbackInputStream`
 - ❑ объединение потоков – `SequenceInputStream`
 - ❑ печатный поток вывода – `PrintStream`
 - ❑ построчное чтение – `LineNumberInputStream` (deprecated)













Произвольный доступ к файлу

- ❑ `RandomAccessFile` используется для осуществления произвольного доступа к содержимому файла:
 - ❑ обеспечивает форматированный ввод-вывод данных (реализует интерфейсы `DataInput` и `DataOutput`)
 - ❑ `void seek(long)` - позиционирование в файле
 - ❑ `long getFilePointer()` - текущая позиция курсора

Символьные потоки (readers & writers)

- В отличие от байтовых потоков, работают на уровне символов, что позволяет корректно работать с различными кодировками
- В качестве мостов между байтовыми и символьными потоками используются классы `InputStreamReader` и `OutputStreamWriter`

Writer
 <code>lock : Object</code>
 <code>Writer()</code>
 <code>Writer(arg0 : Object)</code>
 <code>write(arg0 : int) : void</code>
 <code>write(arg0 : char[]) : void</code>
 <code>write(arg0 : char[], arg1 : int, arg2 : int) : void</code>
 <code>write(arg0 : String) : void</code>
 <code>write(arg0 : String, arg1 : int, arg2 : int) : void</code>
 <code>flush() : void</code>
 <code>close() : void</code>

Reader
 <code>lock : Object</code>
 <code>Reader()</code>
 <code>Reader(arg0 : Object)</code>
 <code>read() : int</code>
 <code>read(arg0 : char[]) : int</code>
 <code>read(arg0 : char[], arg1 : int, arg2 : int) : int</code>
 <code>skip(arg0 : long) : long</code>
 <code>ready() : boolean</code>
 <code>markSupported() : boolean</code>
 <code>mark(arg0 : int) : void</code>
 <code>reset() : void</code>
 <code>close() : void</code>

СИМВОЛЬНЫЕ ПОТОКИ

- Узловые символьные потоки:
 - файловые – `FileReader`, `FileWriter`
 - на основе массивов символов – `CharArrayReader`, `CharArrayWriter`
 - канальные – `PipedReader`, `PipedWriter`
 - строковые – `StringReader`, `StringWriter`
- Фильтрующие символьные потоки:
 - абстрактные – `FilterReader`, `FilterWriter`
 - буферизирующие – `BufferedReader`, `BufferedWriter`
 - переходные – `InputStreamReader`, `OutputStreamWriter`
 - опережающее чтение – `PushbackReader`
 - по строчное чтение – `LineNumberReader`
 - печатный поток вывода – `PrintWriter`

StreamTokenizer class

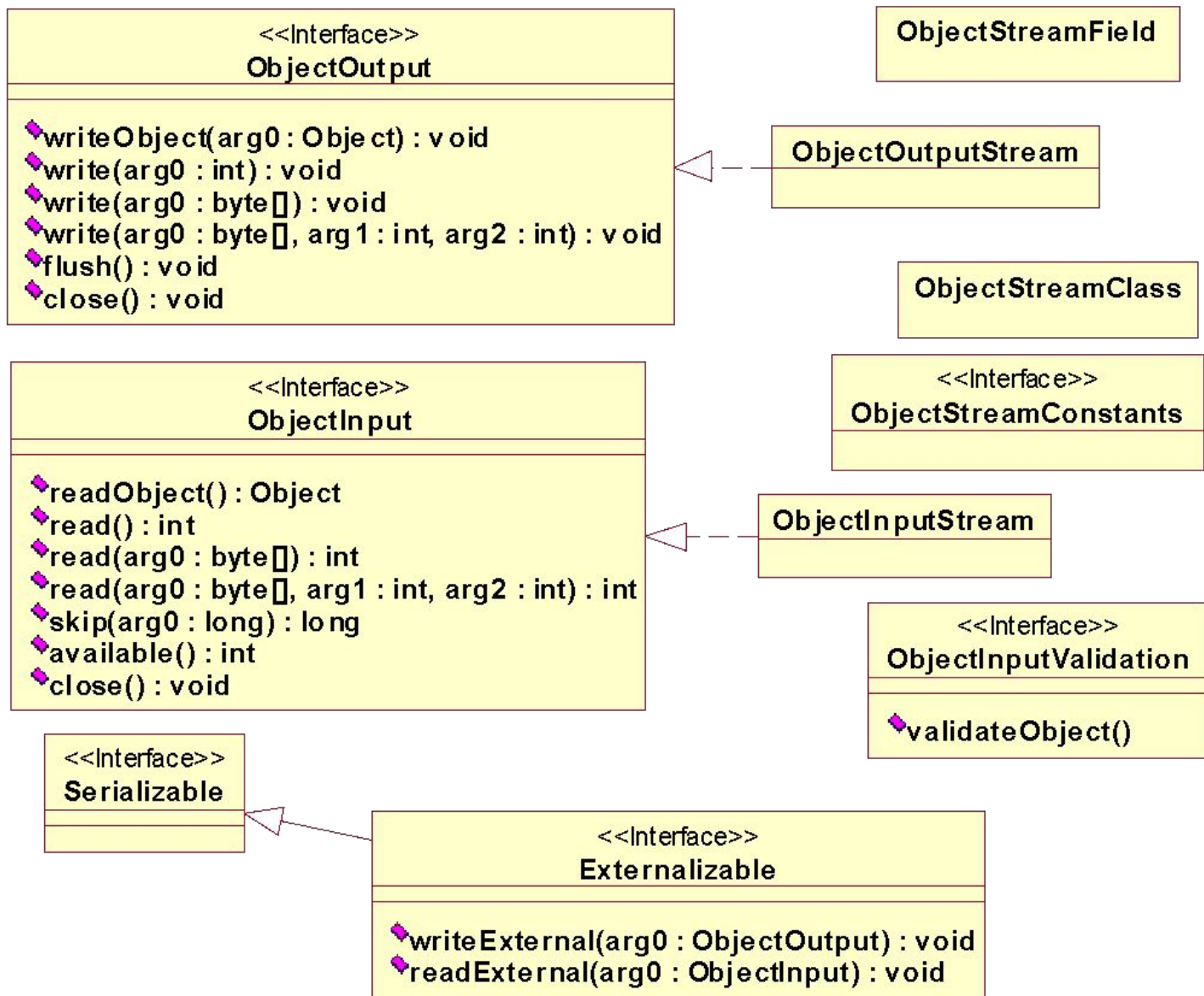
StreamTokenizer

- ◆ **ttype** : int
- ◆ **TT_EOF** : int = -1
- ◆ **TT_EOL** : int = 10
- ◆ **TT_NUMBER** : int = -2
- ◆ **TT_WORD** : int = -3
- ◆ **nval** : double
- ◆ **sval** : String

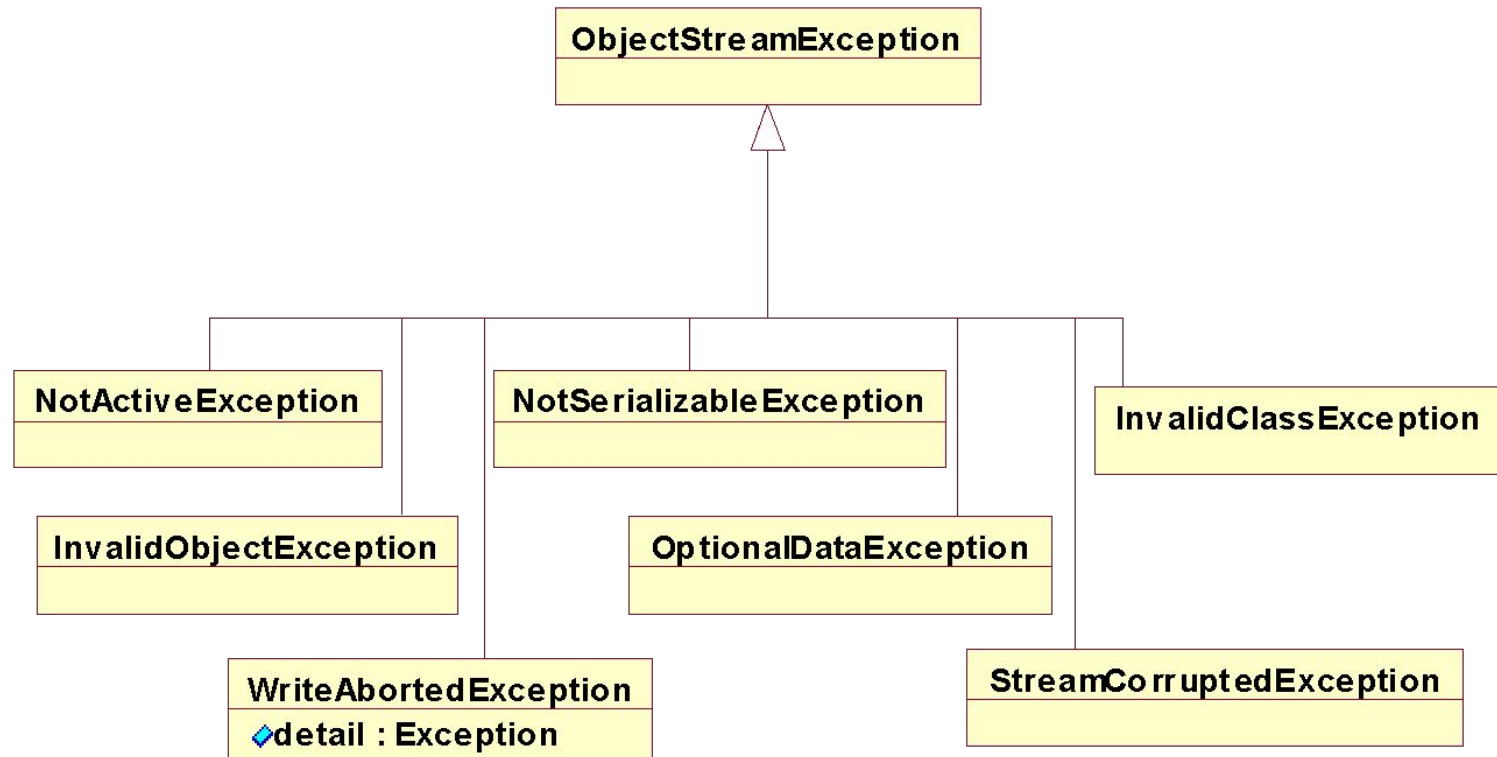
- ◆ **StreamTokenizer(arg0 : InputStream)**
- ◆ **StreamTokenizer(arg0 : Reader)**
- ◆ **resetSyntax() : void**
- ◆ **wordChars(arg0 : int, arg1 : int) : void**
- ◆ **whitespaceChars(arg0 : int, arg1 : int) : void**
- ◆ **ordinaryChars(arg0 : int, arg1 : int) : void**
- ◆ **ordinaryChar(arg0 : int) : void**
- ◆ **commentChar(arg0 : int) : void**
- ◆ **quoteChar(arg0 : int) : void**
- ◆ **parseNumbers() : void**
- ◆ **eollsSignificant(arg0 : boolean) : void**
- ◆ **slashStarComments(arg0 : boolean) : void**
- ◆ **slashSlashComments(arg0 : boolean) : void**
- ◆ **lowerCaseMode(arg0 : boolean) : void**
- ◆ **nextToken() : int**
- ◆ **pushBack() : void**
- ◆ **lineno() : int**
- ◆ **toString() : String**

Сериализация объектов

- ❑ Средства сериализации и десериализации объектов пакета `java.io` реализуют поддержку принципа сохраняемости ООП
- ❑ Сериализация используется для сохранения иерархий объектов для передачи в другую виртуальную машину, либо для последующего восстановления в другом сеансе работы системы
- ❑ Особенности сериализации:
 - ❑ версионность классов сериализуемых объектов
 - ❑ сохранение иерархий
 - ❑ кеширование и переиспользование ссылок на уже сериализованные объекты
 - ❑ средства управления процессом сериализации (десериализации) объектов



Исключения при сериализации



Общие правила

- ❑ Классы, объекты которых подлежат сериализации должны реализовывать маркерный интерфейс `java.io.Serializable`
- ❑ По умолчанию подлежат сохранению все поля объектов не объявленные **transient**. Статические поля не сериализуются
- ❑ Для сериализации и десериализации используются фильтрующие потоки `ObjectOutputStream` и `ObjectInputStream`
- ❑ Сериализуемые поля класса реализующего интерфейс `Serializable` должны быть примитивными типами либо типами, реализующими интерфейс `Serializable`
- ❑ Если класс имеет своим суперклассом класс, не реализующий интерфейс `Serializable`, то у данного суперкласса должен быть доступен конструктор с пустым списком аргументов
- ❑ Сериализации внутренних или анонимных классов следует категорически избегать

Управление сериализацией объекта

- Для явного определения списка сериализуемых полей нужно в классе объявить поле `private static final ObjectOutputStreamField[] serialPersistentFields = {...};`
- Для управления процессом сериализации/десериализации нужно в классе объявить методы:
 - `private void writeObject(java.io.ObjectOutputStream out) throws IOException {...}`
 - `private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException {...}`
- Для управления замещением объекта при сериализации/десериализации нужно определить методы:
 - `Object writeReplace() throws ObjectOutputStreamException {...}`
 - `Object readResolve() throws ObjectOutputStreamException {...}`
- Для явного задания номера версии нужно определить поле
 - `static final long serialVersionUID = ...;`

Интерфейс `java.io.Externalizable`

- Позволяет полностью управлять форматом записи данных класса (автоматически сохраняется только информация об классе и объекте)
- Для сохранения состояния нужно реализовать метод
 - `public void writeExternal(ObjectOutput out) throws IOException`
- Для восстановления состояния нужно реализовать метод
 - `public void readExternal(ObjectInput in) throws IOException, java.lang.ClassNotFoundException`
- Класс реализующий интерфейс `Externalizable` должен иметь `public` конструктор с пустым списком аргументов

Версионность

- Изменения, ведущие к несовместимости версий классов:
 - Удаление полей не объявленных `transient`
 - Изменение позиции класса в иерархии классов
 - Изменение факта наличия/отсутствия модификаторов `static` и `transient` у поля
 - Изменение типа у поля примитивного (встроенного) типа
 - Изменение реализации методов `writeObject()` и `readObject()` на предмет использования методов `defaultWriteObject()` и `defaultReadObject()` классов `ObjectOutputStream` и `ObjectInputStream` соответственно
 - Изменение класса с `Serializable` на `Externalizable` и наоборот
 - Добавление методов `writeReplace()` или `readResolve()`
 - Изменение факта наследования классом интерфейсов `Serializable` или `Externalizable`

Работа с сетью

- Для работы с сетью используется пакет `java.net`, предоставляющий средства:
 - адресации в Internet (`InetAddress`, + средства `java 1.4`: `Inet4Address`, `Inet6Address`, `InetSocketAddress`, `SocketAddress`)
 - работы с протоколом TCP (`Socket`, `ServerSocket`, `SocketOptions`)
 - работы с протоколом UDP (`DatagramSocket`, `DatagramPacket`, `MulticastSocket`)
 - работы с URL и поддержки HTTP (`URL`, `URLConnection`, `HttpURLConnection`, `JarURLConnection`, `URLDecoder`, `URLEncoder`, `ContentHandler`, `URLStreamHandler`)
 - авторизации в Internet (`Authenticator`, `PasswordAuthentication`)
 - динамической загрузки классов из сети (`URLClassLoader`)

Исключения пакета java.net

