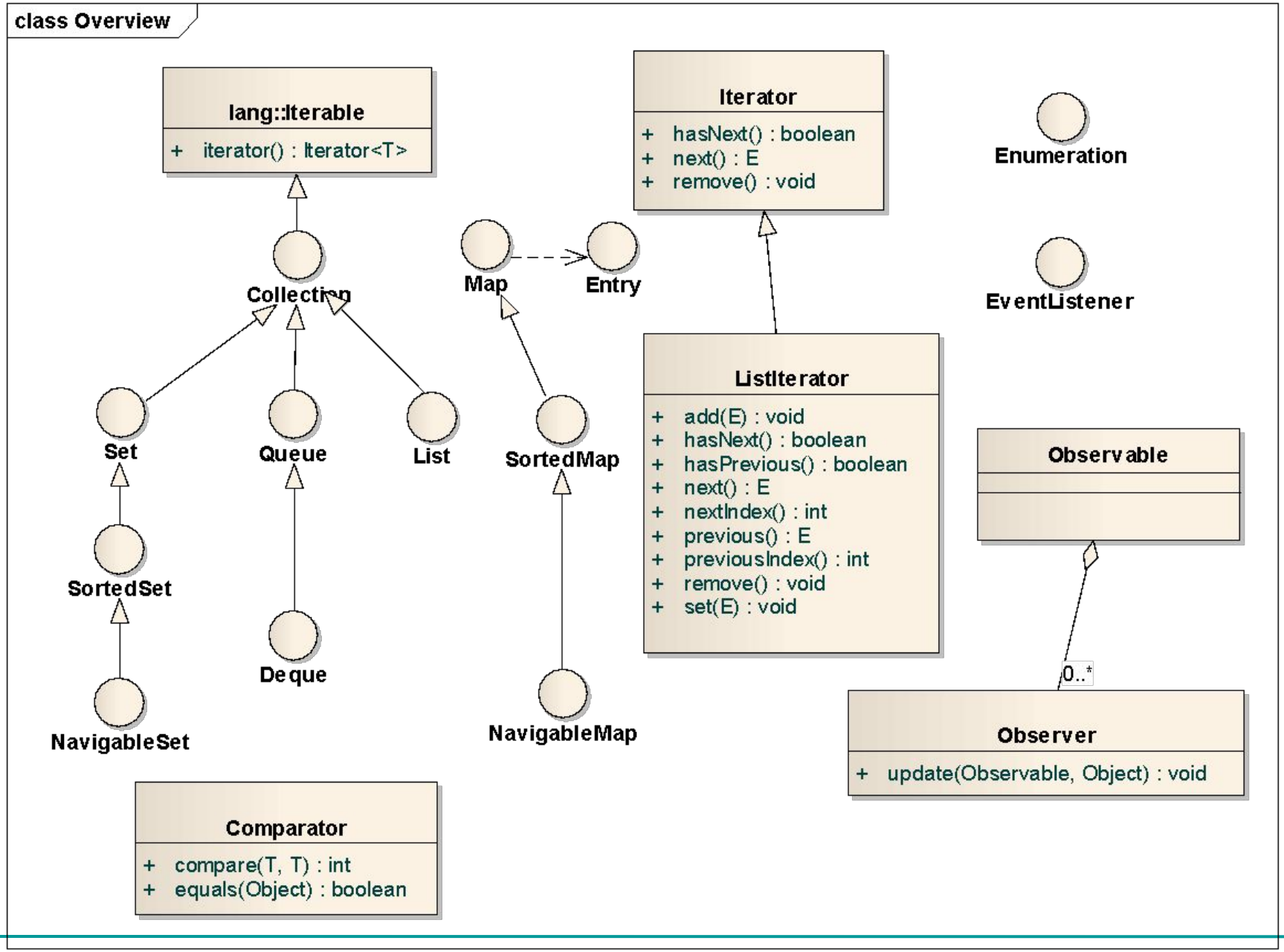
# Основы программирования на языке Java

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


Новосибирск, 2004


class Overview
lang::Iterable
+ iterator() : Iterator<T>
Iterator
+ hasNext() : boolean
+ next() : E
+ remove() : void
Enumeration
EventListener
Collection
Map
Entry
Set
Queue
List
SortedMap
ListIterator
+ add(E) : void
+ hasNext() : boolean
+ hasPrevious() : boolean
+ next() : E
+ nextIndex() : int
+ previous() : E
+ previousIndex() : int
+ remove() : void
+ set(E) : void
Observable
SortedSet
Deque
NavigableSet
NavigableMap
0..*
Observer
+ update(Observable, Object) : void
Comparator
+ compare(T, T) : int
+ equals(Object) : boolean

# Enumeration, Comparator

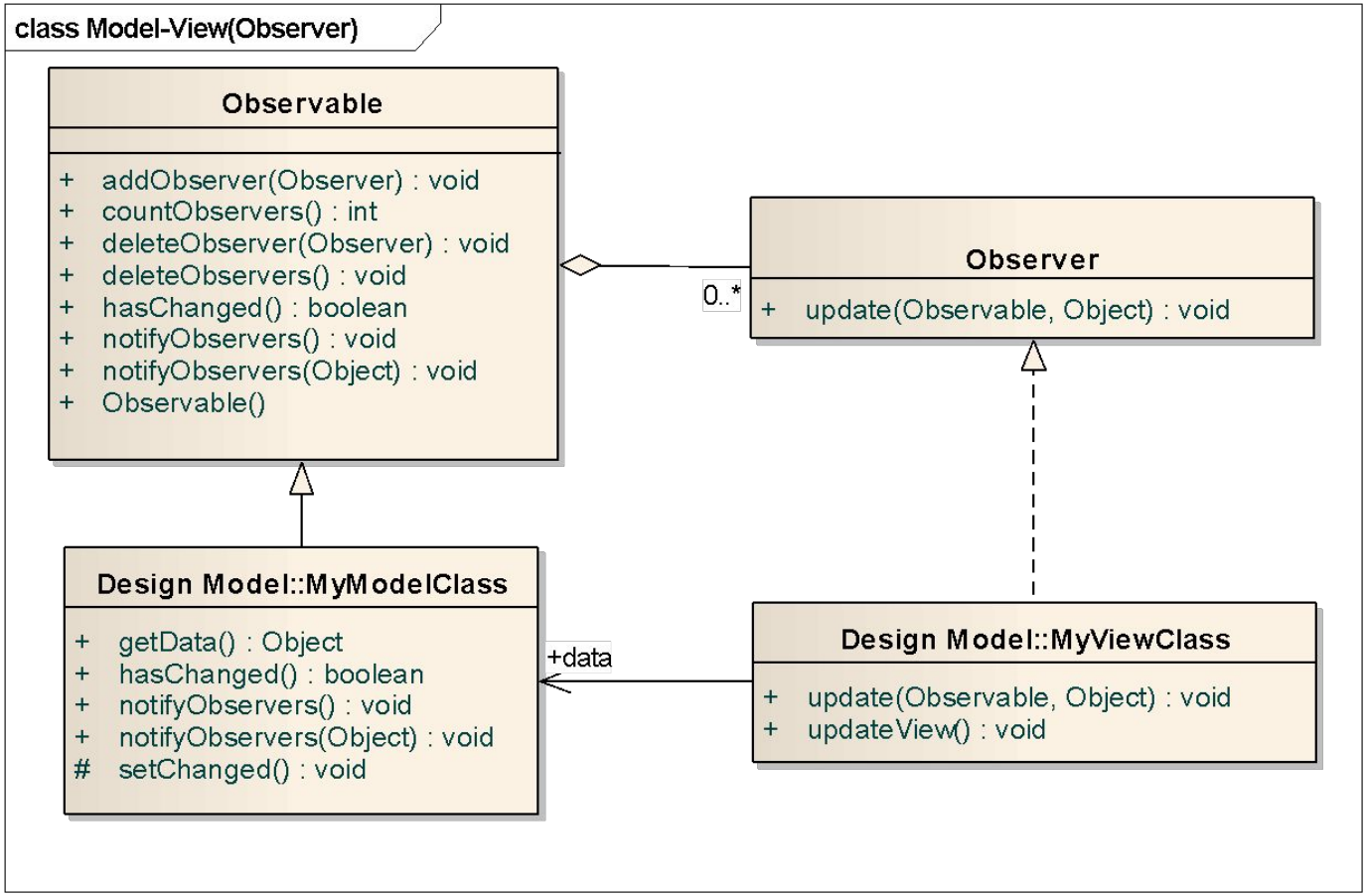
- Enumeration – объекты классов, реализующих данный интерфейс, используются для предоставления однопроходного последовательного доступа к серии объектов:

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

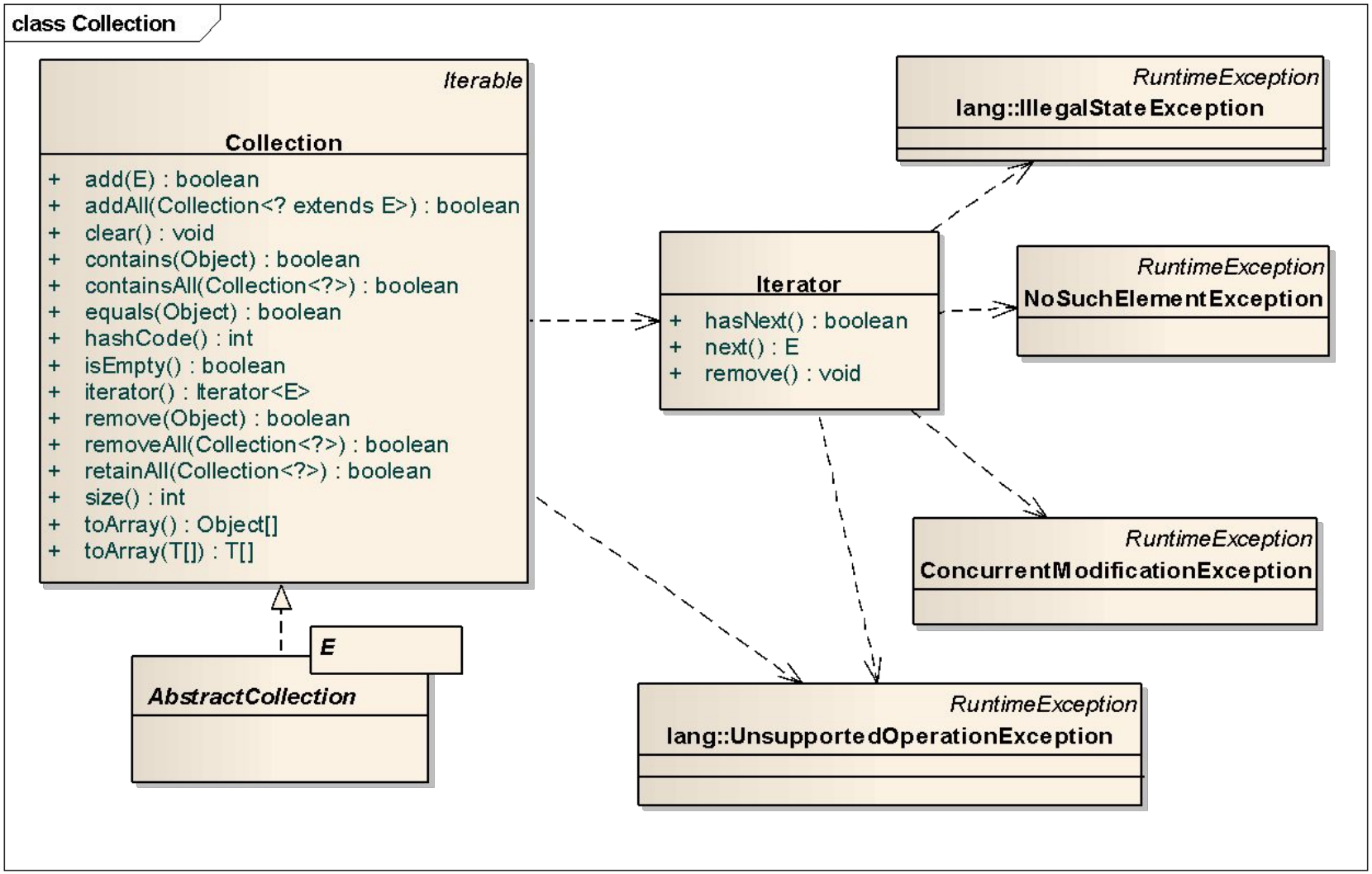
- Comparator – классы, реализующие данный интерфейс используются для обеспечения сравнения произвольных элементов. Используются при сортировке и организации упорядоченных контейнеров:

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

# Паттерн Model-View (Observer)



# Коллекция объектов (контейнер)

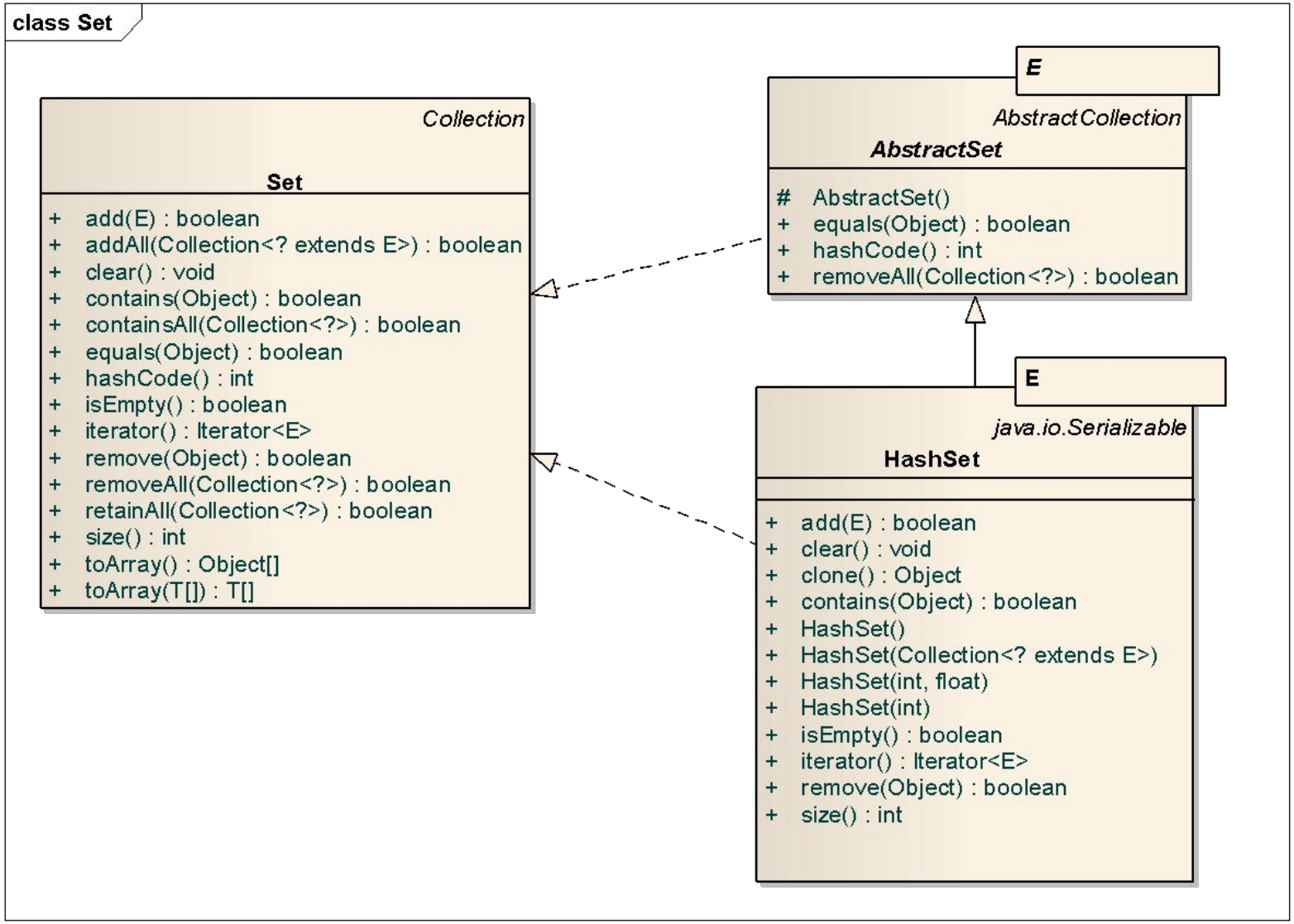


# Использование итераторов

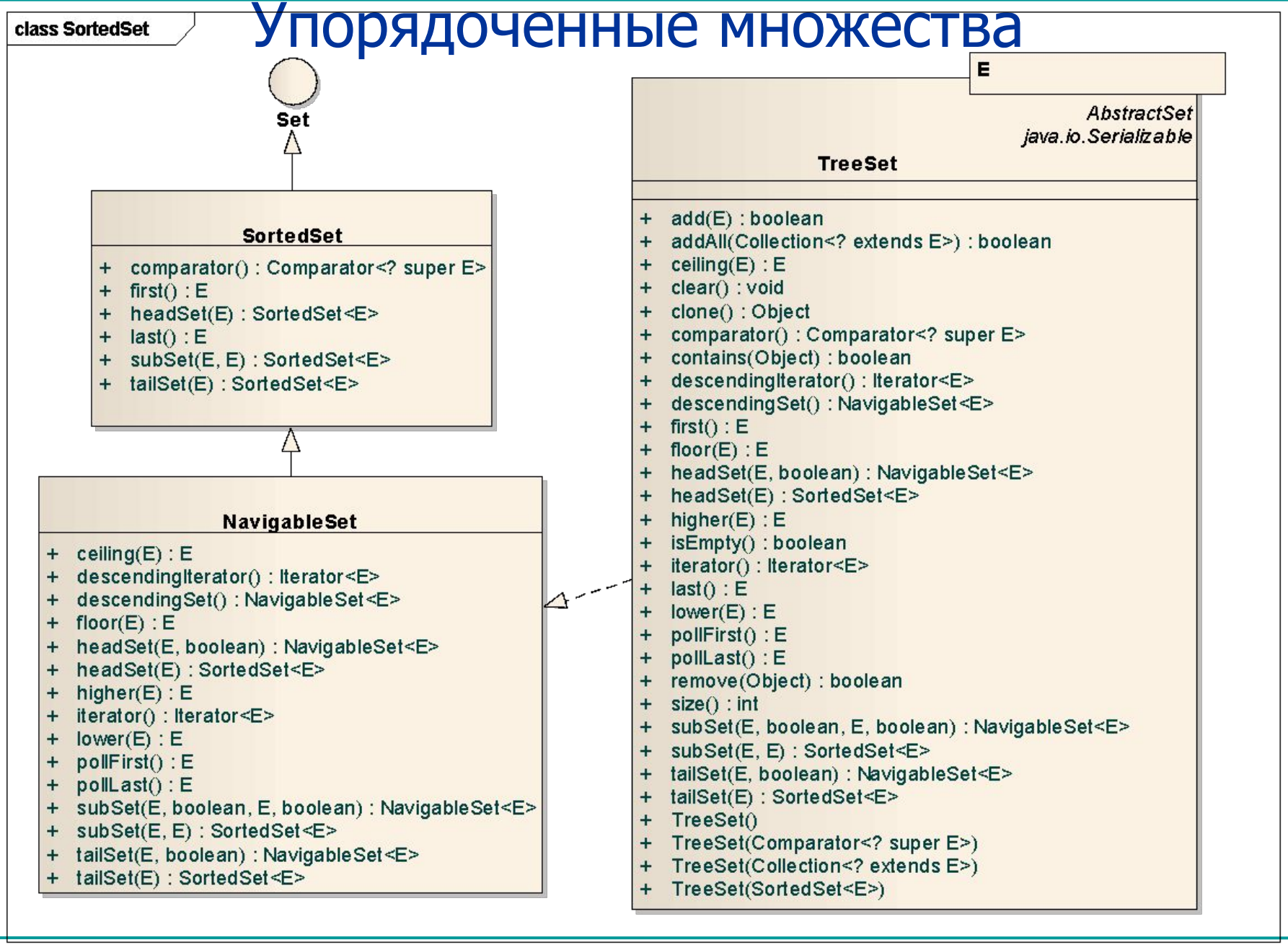
```
Collection<String> col = …;
for (Iterator<String> i = col.iterator(); i.hasNext(); ) {
    String s = i.next();
}
for (String s : col ) { s; } //Альтернатива
```

- 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 flase
- UnsupportedOperationException – возникает при вызове метода который не поддерживается конкретной реализацией коллекции

# Множества

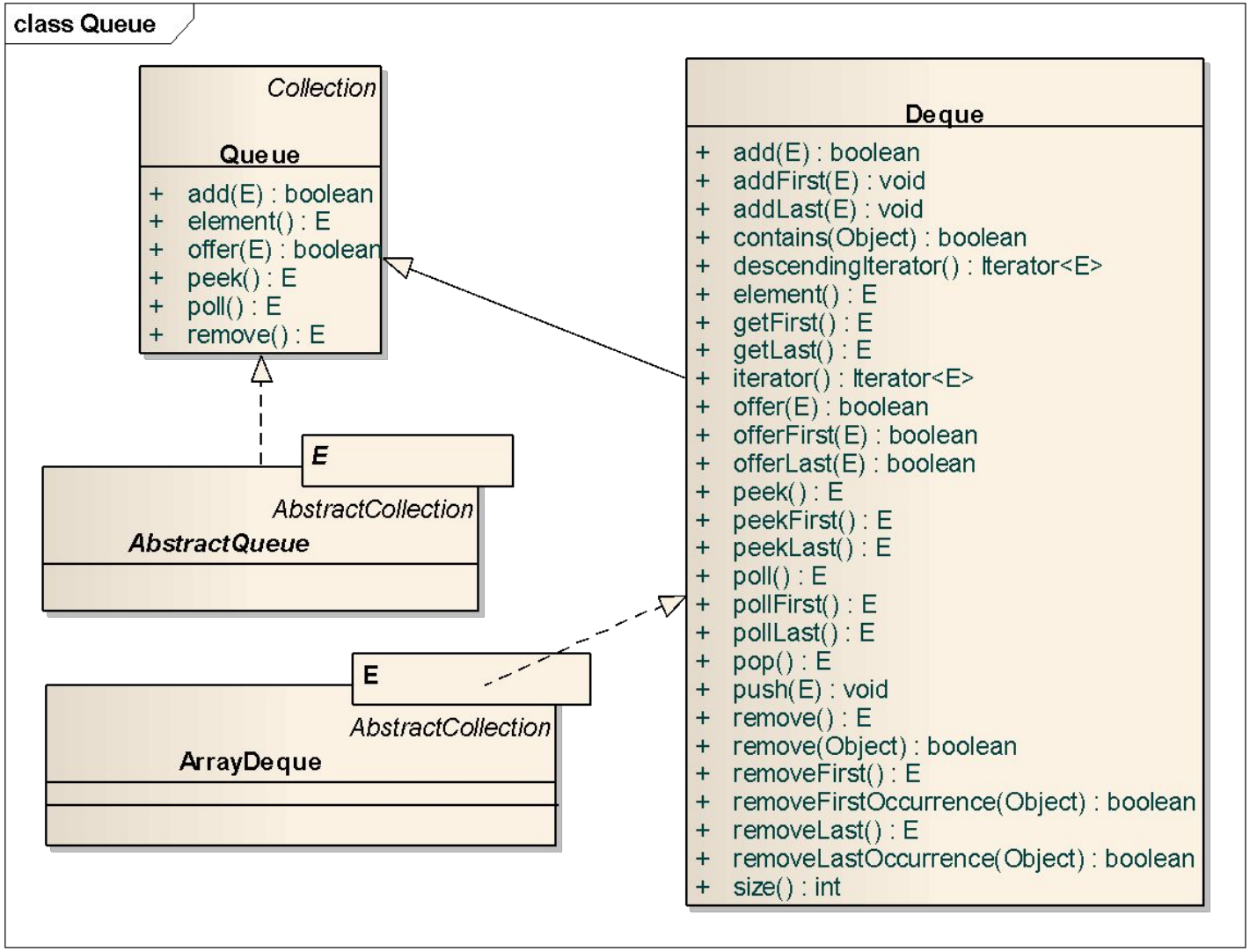

class Set
Collection
Set
+ add(E) : boolean
+ addAll(Collection<? extends E>) : boolean
+ clear() : void
+ contains(Object) : boolean
+ containsAll(Collection<?>) : boolean
+ equals(Object) : boolean
+ hashCode() : int
+ isEmpty() : boolean
+ iterator() : Iterator<E>
+ remove(Object) : boolean
+ removeAll(Collection<?>) : boolean
+ retainAll(Collection<?>) : boolean
+ size() : int
+ toArray() : Object[]
+ toArray(T[]) : T[]
E
AbstractCollection
AbstractSet
# AbstractSet()
+ equals(Object) : boolean
+ hashCode() : int
+ removeAll(Collection<?>) : boolean
E
java.io.Serializable
HashSet
+ add(E) : boolean
+ clear() : void
+ clone() : Object
+ contains(Object) : boolean
+ HashSet()
+ HashSet(Collection<? extends E>)
+ HashSet(int, float)
+ HashSet(int)
+ isEmpty() : boolean
+ iterator() : Iterator<E>
+ remove(Object) : boolean
+ size() : int

# Упорядоченные множества

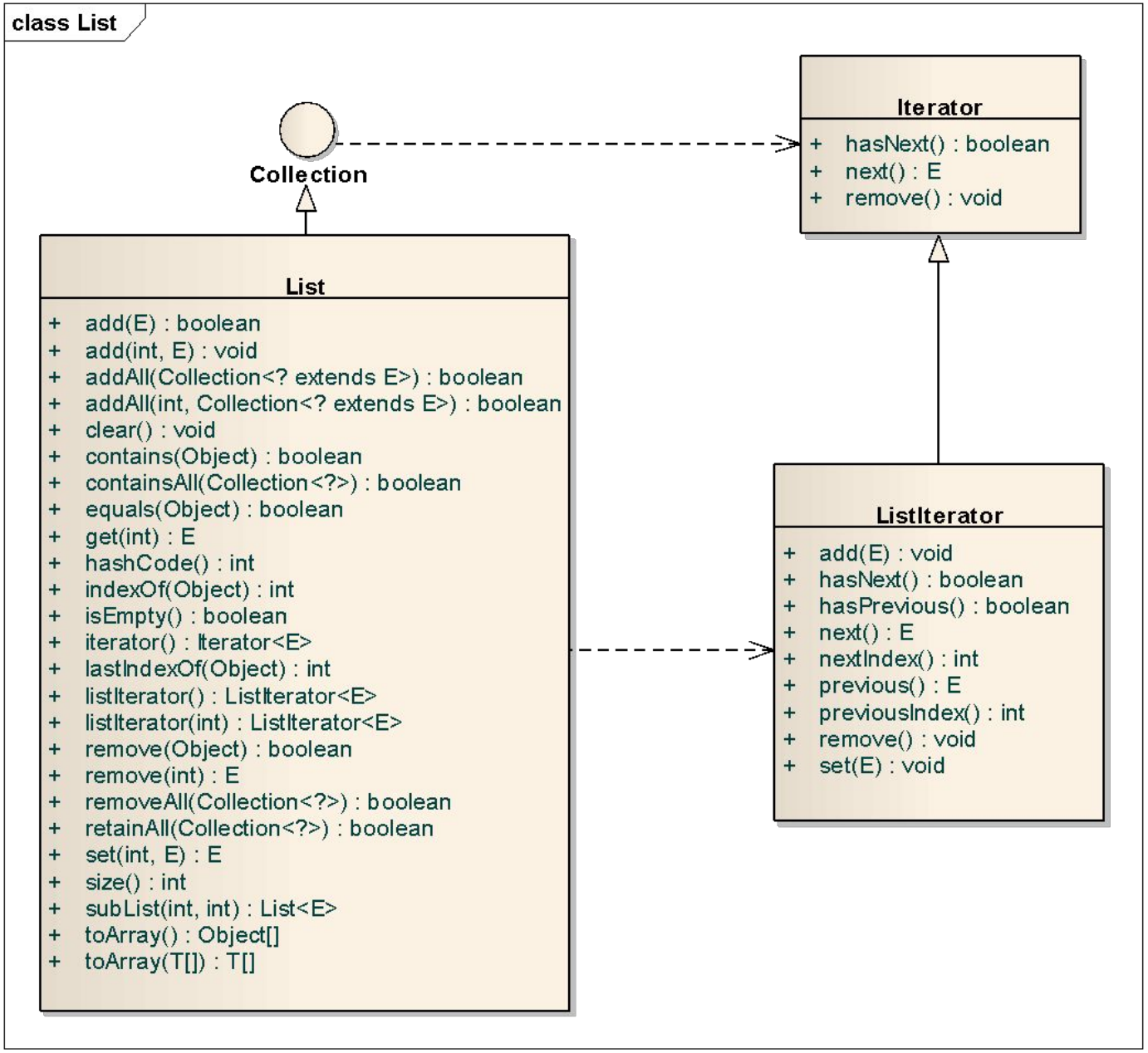


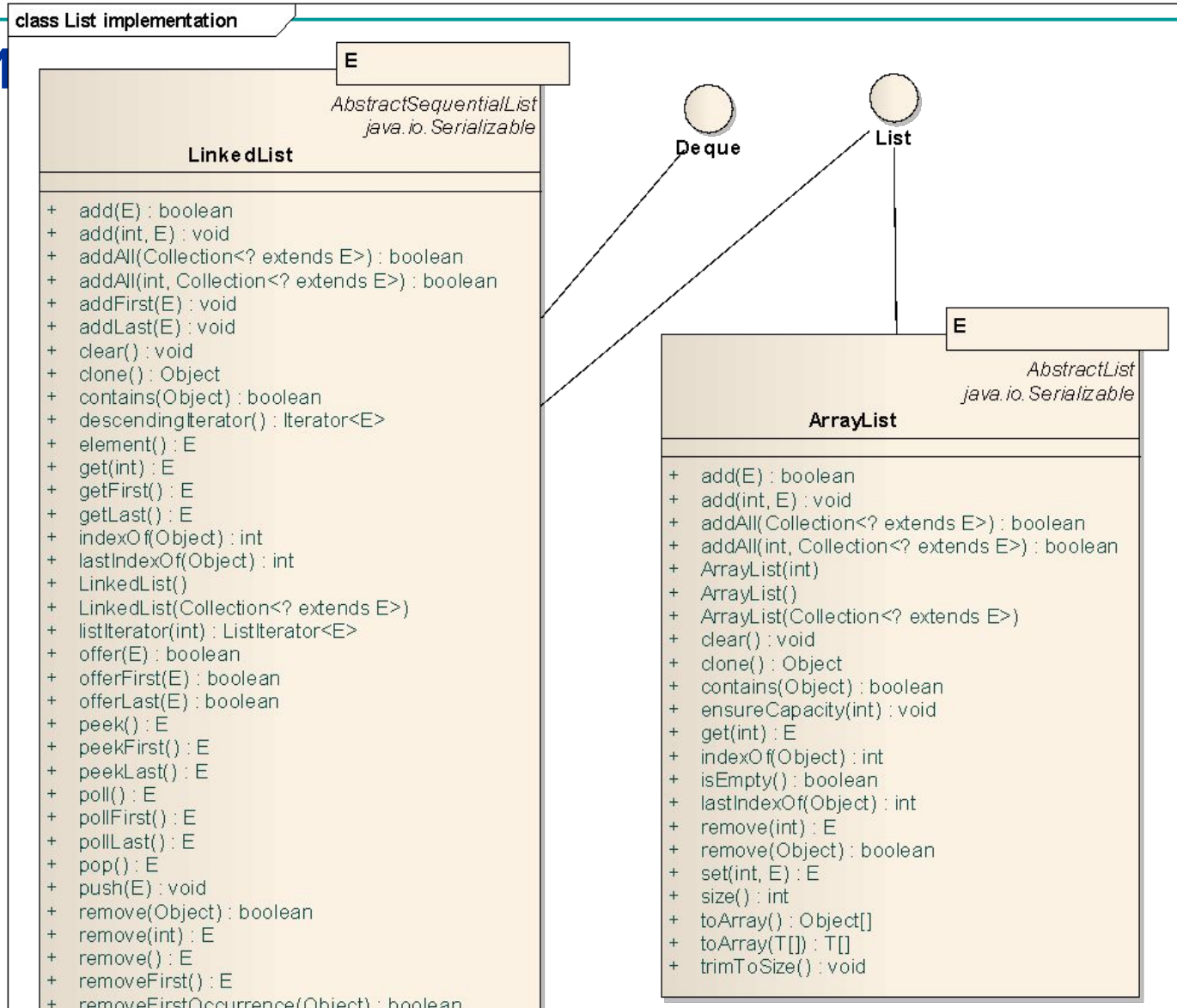
# Очереди


class Queue
Collection
Queue
+ add(E) : boolean
+ element() : E
+ offer(E) : boolean
+ peek() : E
+ poll() : E
+ remove() : E
E
AbstractCollection
AbstractQueue
E
AbstractCollection
ArrayDeque
Deque
+ add(E) : boolean
+ addFirst(E) : void
+ addLast(E) : void
+ contains(Object) : boolean
+ descendingIterator() : Iterator<E>
+ element() : E
+ getFirst() : E
+ getLast() : E
+ iterator() : Iterator<E>
+ offer(E) : boolean
+ offerFirst(E) : boolean
+ offerLast(E) : boolean
+ peek() : E
+ peekFirst() : E
+ peekLast() : E
+ poll() : E
+ pollFirst() : E
+ pollLast() : E
+ pop() : E
+ push(E) : void
+ remove() : E
+ remove(Object) : boolean
+ removeFirst() : E
+ removeFirstOccurrence(Object) : boolean
+ removeLast() : E
+ removeLastOccurrence(Object) : boolean
+ size() : int

# Списки


class List
Collection
Iterator
+ hasNext() : boolean
+ next() : E
+ remove() : void
List
+ add(E) : boolean
+ add(int, E) : void
+ addAll(Collection<? extends E>) : boolean
+ addAll(int, Collection<? extends E>) : boolean
+ clear() : void
+ contains(Object) : boolean
+ containsAll(Collection<?>) : boolean
+ equals(Object) : boolean
+ get(int) : E
+ hashCode() : int
+ indexOf(Object) : int
+ isEmpty() : boolean
+ iterator() : Iterator<E>
+ lastIndexOf(Object) : int
+ listIterator() : ListIterator<E>
+ listIterator(int) : ListIterator<E>
+ remove(Object) : boolean
+ remove(int) : E
+ removeAll(Collection<?>) : boolean
+ retainAll(Collection<?>) : boolean
+ set(int, E) : E
+ size() : int
+ subList(int, int) : List<E>
+ toArray() : Object[]
+ toArray(T[]) : T[]
ListIterator
+ add(E) : void
+ hasNext() : boolean
+ hasPrevious() : boolean
+ next() : E
+ nextIndex() : int
+ previous() : E
+ previousIndex() : int
+ remove() : void
+ set(E) : void

# Реализации списка

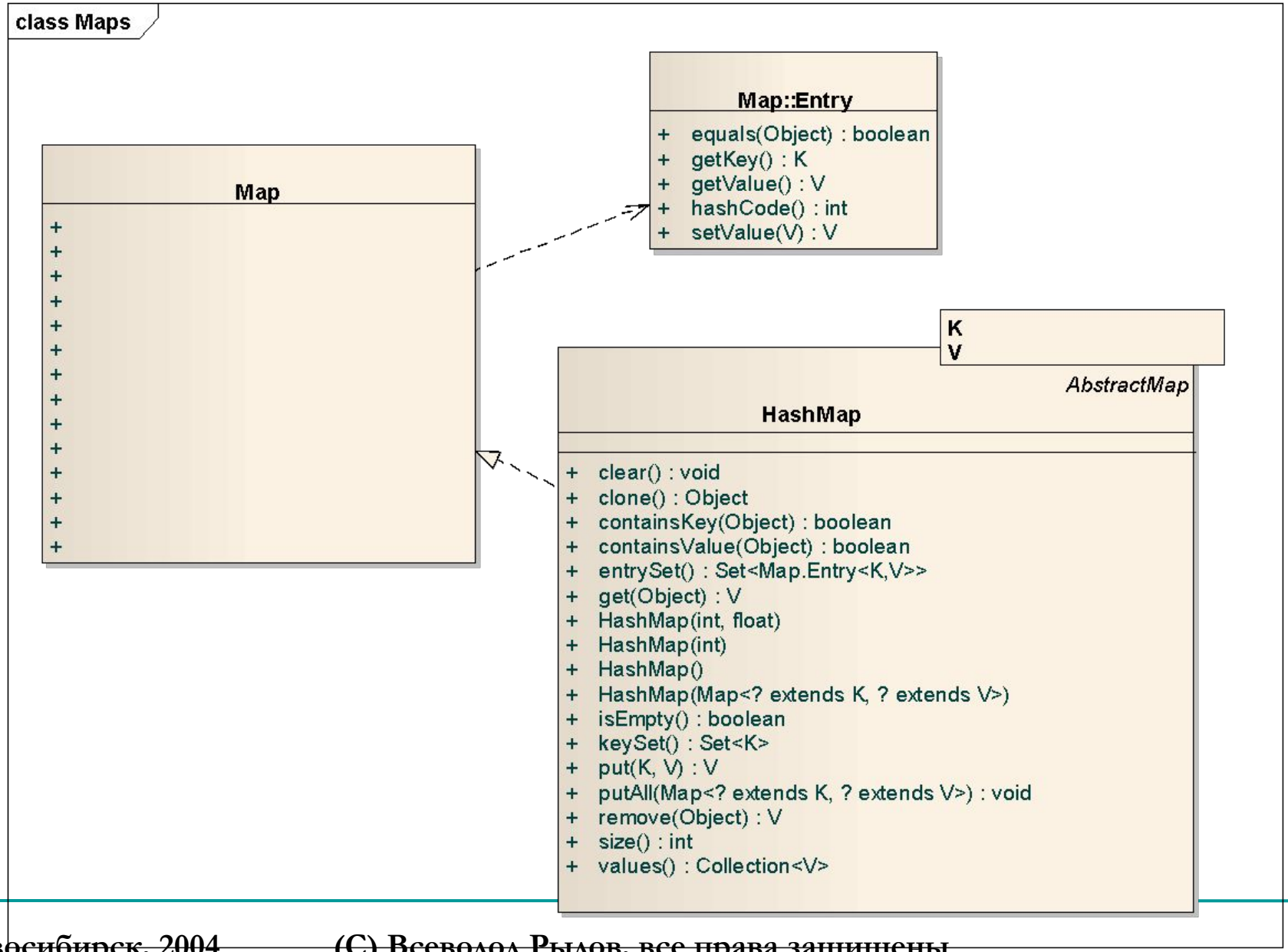


# Пример простой реализации стека

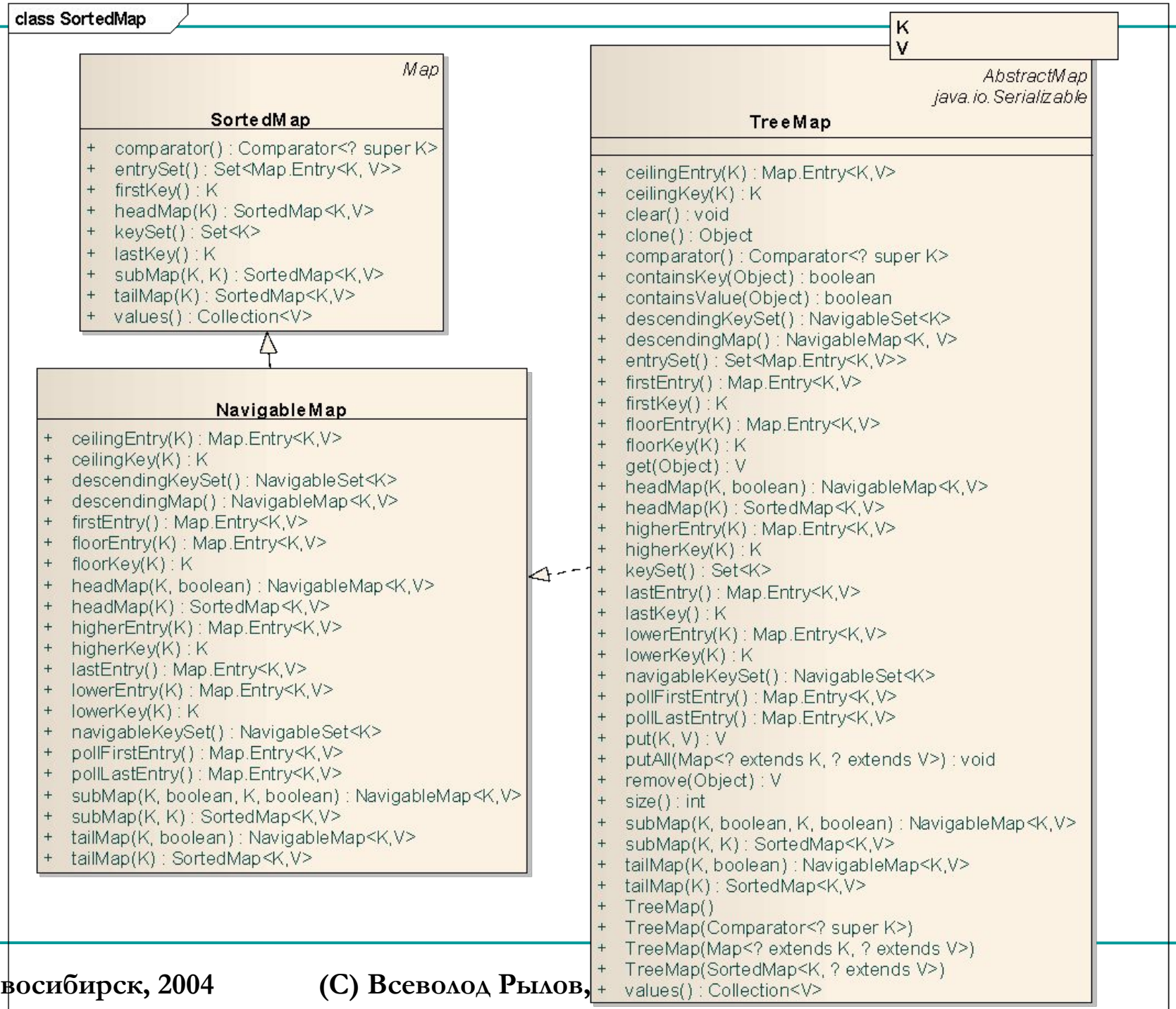
```
import java.util.*;
public class Stack<T> {
    private LinkedList<T> content;
    public Stack()  {
        content = new LinkedList<T>();
    }
    public void push(T obj)  {
        content.addFirst(obj);
    }
    public T pop() throws
     NoSuchElementException
    {
       return content.removeFirst();
    }
    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());
      }
    }
}
```

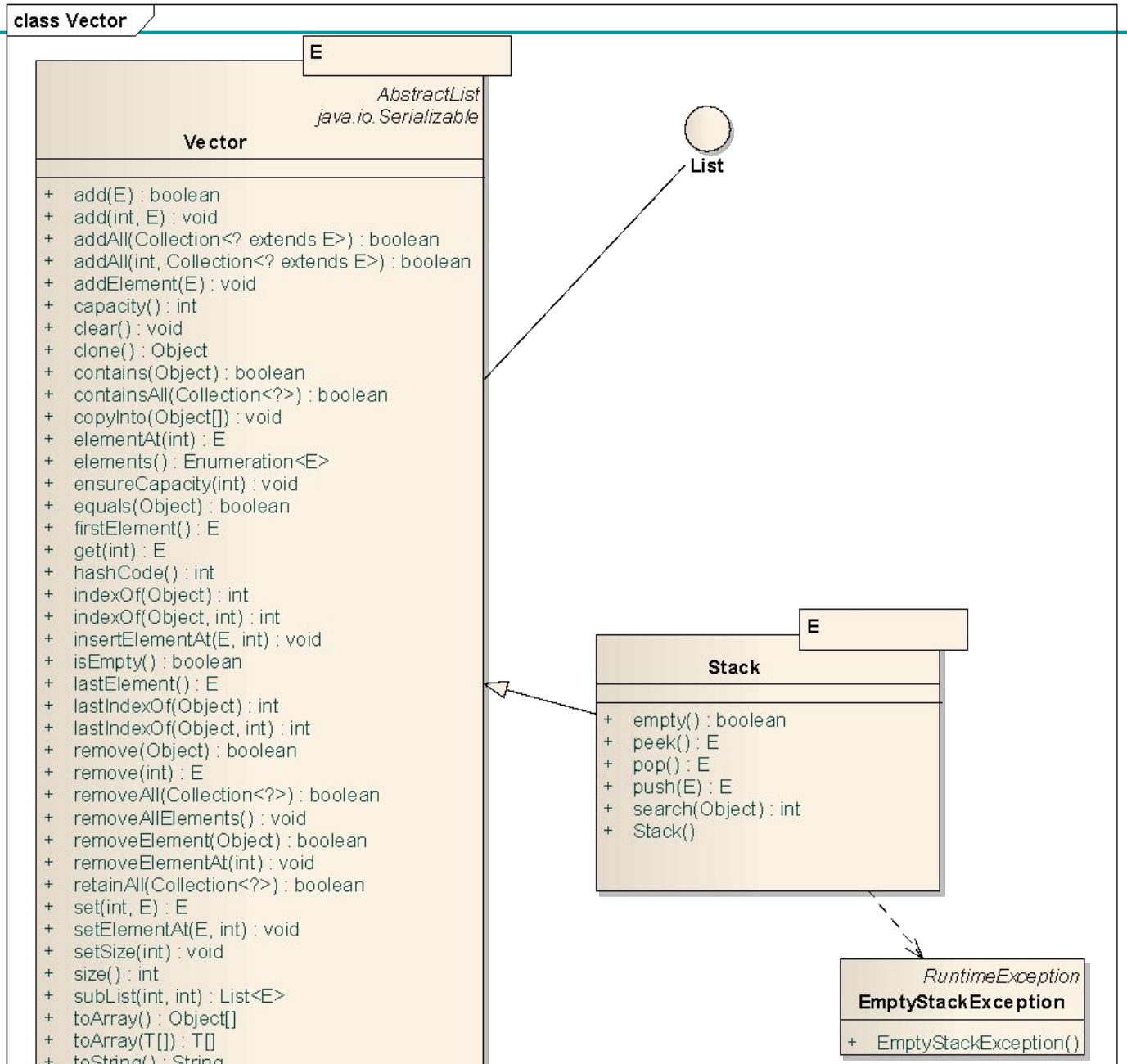
# Ассоциативные контейнеры



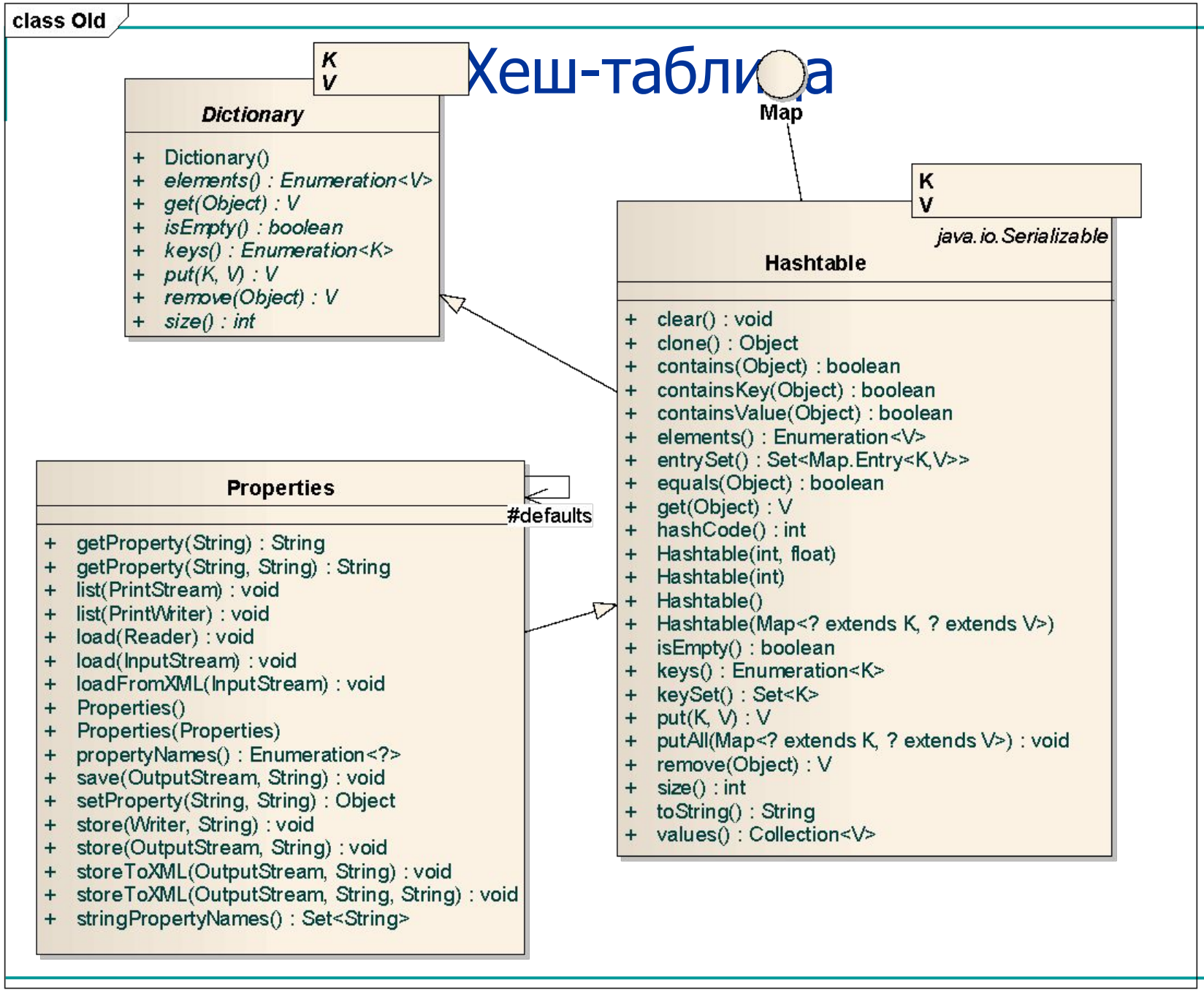

  


class SortedMap
Map
SortedMap
+ comparator() : Comparator<? super K>
+ entrySet() : Set<Map.Entry<K, V>>
+ firstKey() : K
+ headMap(K) : SortedMap<K,V>
+ keySet() : Set<K>
+ lastKey() : K
+ subMap(K, K) : SortedMap<K,V>
+ tailMap(K) : SortedMap<K,V>
+ values() : Collection<V>
NavigableMap
+ ceilingEntry(K) : Map.Entry<K,V>
+ ceilingKey(K) : K
+ descendingKeySet() : NavigableSet<K>
+ descendingMap() : NavigableMap<K,V>
+ firstEntry() : Map.Entry<K,V>
+ floorEntry(K) : Map.Entry<K,V>
+ floorKey(K) : K
+ headMap(K, boolean) : NavigableMap<K,V>
+ headMap(K) : SortedMap<K,V>
+ higherEntry(K) : Map.Entry<K,V>
+ higherKey(K) : K
+ lastEntry() : Map.Entry<K,V>
+ lowerEntry(K) : Map.Entry<K,V>
+ lowerKey(K) : K
+ navigableKeySet() : NavigableSet<K>
+ pollFirstEntry() : Map.Entry<K,V>
+ pollLastEntry() : Map.Entry<K,V>
+ subMap(K, boolean, K, boolean) : NavigableMap<K,V>
+ subMap(K, K) : SortedMap<K,V>
+ tailMap(K, boolean) : NavigableMap<K,V>
+ tailMap(K) : SortedMap<K,V>
K
V
AbstractMap
java.io.Serializable
TreeMap
+ ceilingEntry(K) : Map.Entry<K,V>
+ ceilingKey(K) : K
+ clear() : void
+ clone() : Object
+ comparator() : Comparator<? super K>
+ containsKey(Object) : boolean
+ containsValue(Object) : boolean
+ descendingKeySet() : NavigableSet<K>
+ descendingMap() : NavigableMap<K, V>
+ entrySet() : Set<Map.Entry<K,V>>
+ firstEntry() : Map.Entry<K,V>
+ firstKey() : K
+ floorEntry(K) : Map.Entry<K,V>
+ floorKey(K) : K
+ get(Object) : V
+ headMap(K, boolean) : NavigableMap<K,V>
+ headMap(K) : SortedMap<K,V>
+ higherEntry(K) : Map.Entry<K,V>
+ higherKey(K) : K
+ keySet() : Set<K>
+ lastEntry() : Map.Entry<K,V>
+ lastKey() : K
+ lowerEntry(K) : Map.Entry<K,V>
+ lowerKey(K) : K
+ navigableKeySet() : NavigableSet<K>
+ pollFirstEntry() : Map.Entry<K,V>
+ pollLastEntry() : Map.Entry<K,V>
+ put(K, V) : V
+ putAll(Map<? extends K, ? extends V>) : void
+ remove(Object) : V
+ size() : int
+ subMap(K, boolean, K, boolean) : NavigableMap<K,V>
+ subMap(K, K) : SortedMap<K,V>
+ tailMap(K, boolean) : NavigableMap<K,V>
+ tailMap(K) : SortedMap<K,V>
+ TreeMap()
+ TreeMap(Comparator<? super K>)
+ TreeMap(Map<? extends K, ? extends V>)
+ TreeMap(SortedMap<K, ? extends V>)
+ values() : Collection<V>

# Вектор



# Хеш-таблица


class Old
K
V
Dictionary
+ Dictionary()
+ elements() : Enumeration<V>
+ get(Object) : V
+ isEmpty() : boolean
+ keys() : Enumeration<K>
+ put(K, V) : V
+ remove(Object) : V
+ size() : int
Map
K
V
java.io.Serializable
Hashtable
+ clear() : void
+ clone() : Object
+ contains(Object) : boolean
+ containsKey(Object) : boolean
+ containsValue(Object) : boolean
+ elements() : Enumeration<V>
+ entrySet() : Set<Map.Entry<K,V>>
+ equals(Object) : boolean
+ get(Object) : V
+ hashCode() : int
+ Hashtable(int, float)
+ Hashtable(int)
+ Hashtable()
+ Hashtable(Map<? extends K, ? extends V>)
+ isEmpty() : boolean
+ keys() : Enumeration<K>
+ keySet() : Set<K>
+ put(K, V) : V
+ putAll(Map<? extends K, ? extends V>) : void
+ remove(Object) : V
+ size() : int
+ toString() : String
+ values() : Collection<V>
Properties
#defaults
+ getProperty(String) : String
+ getProperty(String, String) : String
+ list(PrintStream) : void
+ list(PrintWriter) : void
+ load(Reader) : void
+ load(InputStream) : void
+ loadFromXML(InputStream) : void
+ Properties()
+ Properties(Properties)
+ propertyNames() : Enumeration<?>
+ save(OutputStream, String) : void
+ setProperty(String, String) : Object
+ store(Writer, String) : void
+ store(OutputStream, String) : void
+ storeToXML(OutputStream, String) : void
+ storeToXML(OutputStream, String, String) : void
+ stringPropertyNames() : Set<String>

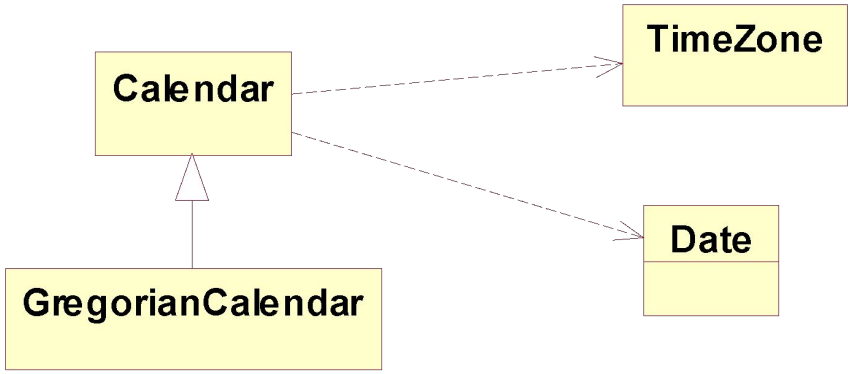
# Класс-утилита Collections

- 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

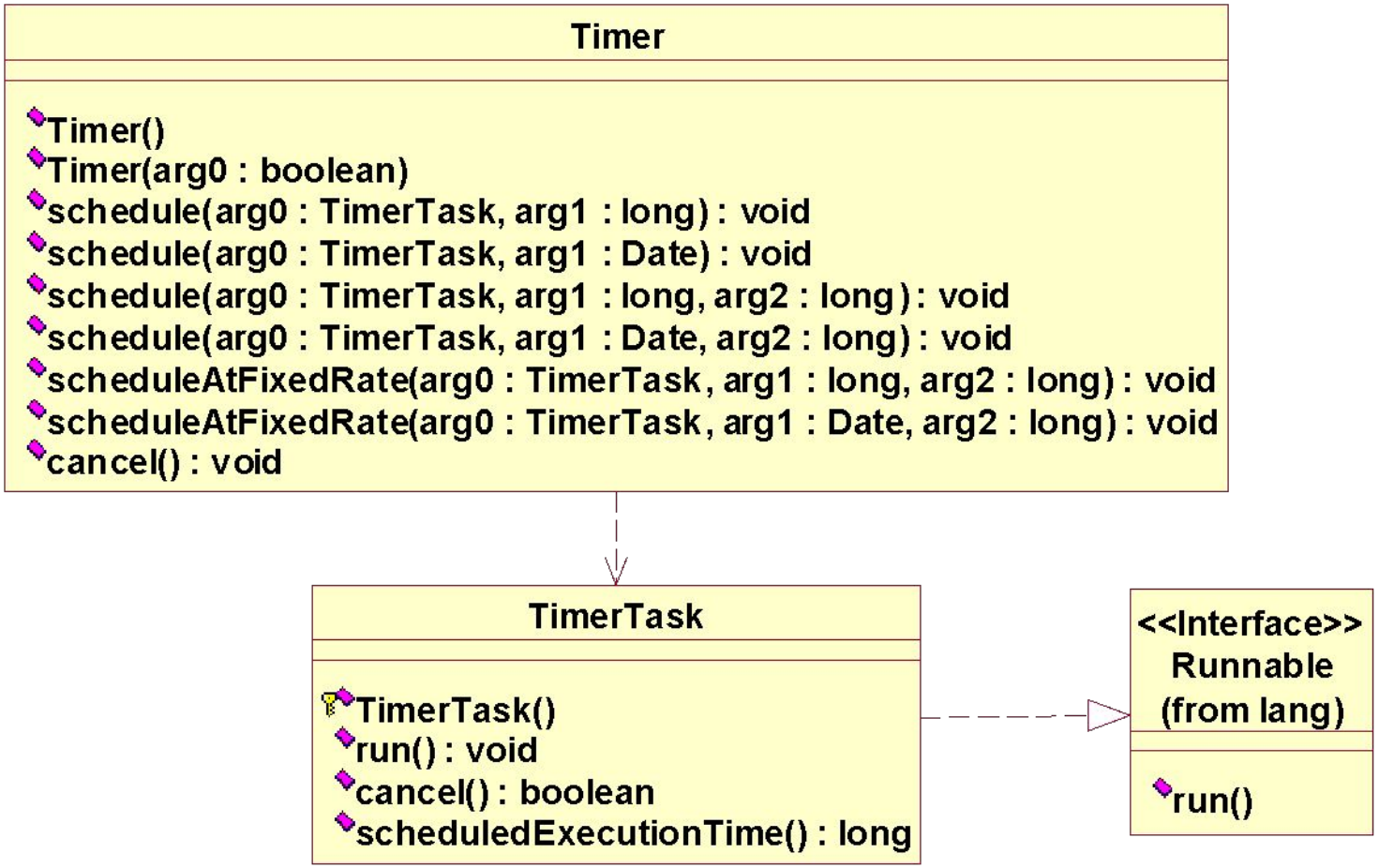
# Класс-утилита Collections (продолж.)

- 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

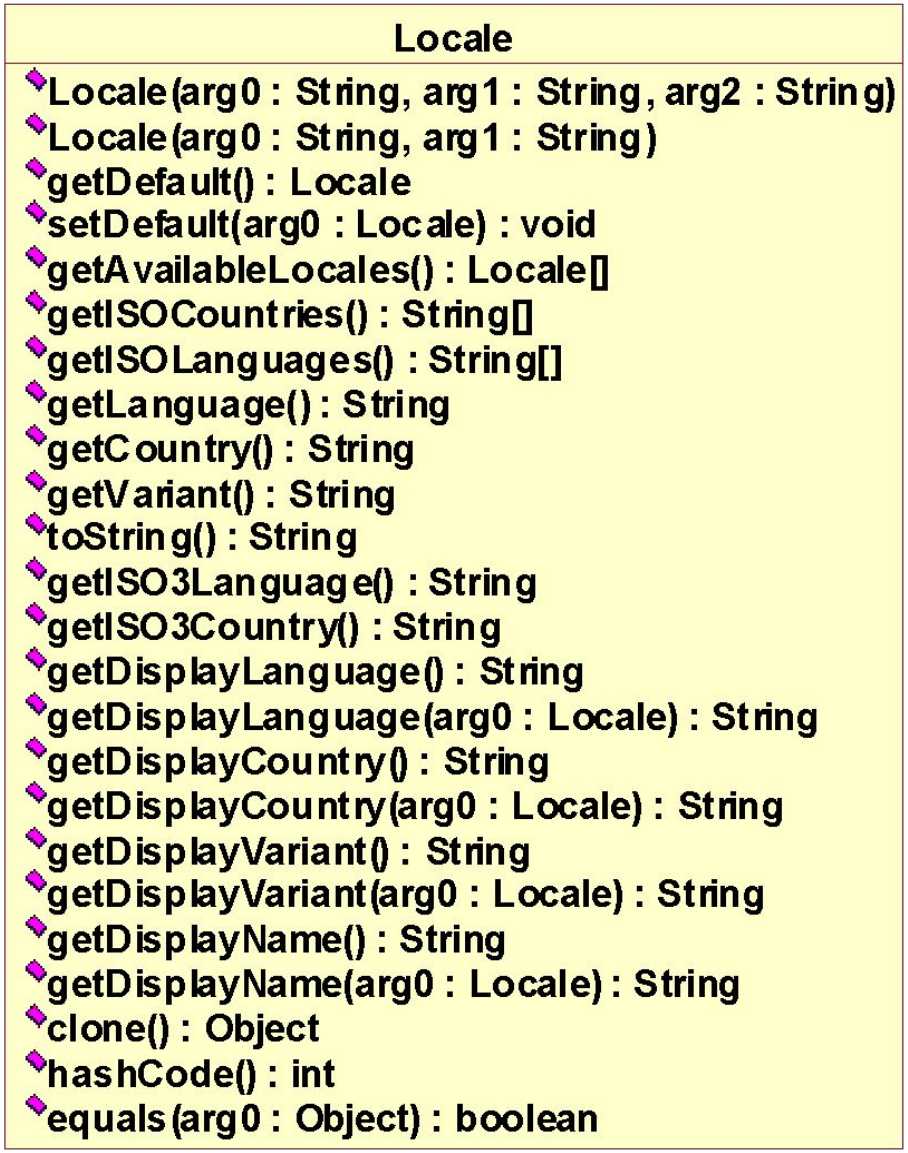
# Работа со временем



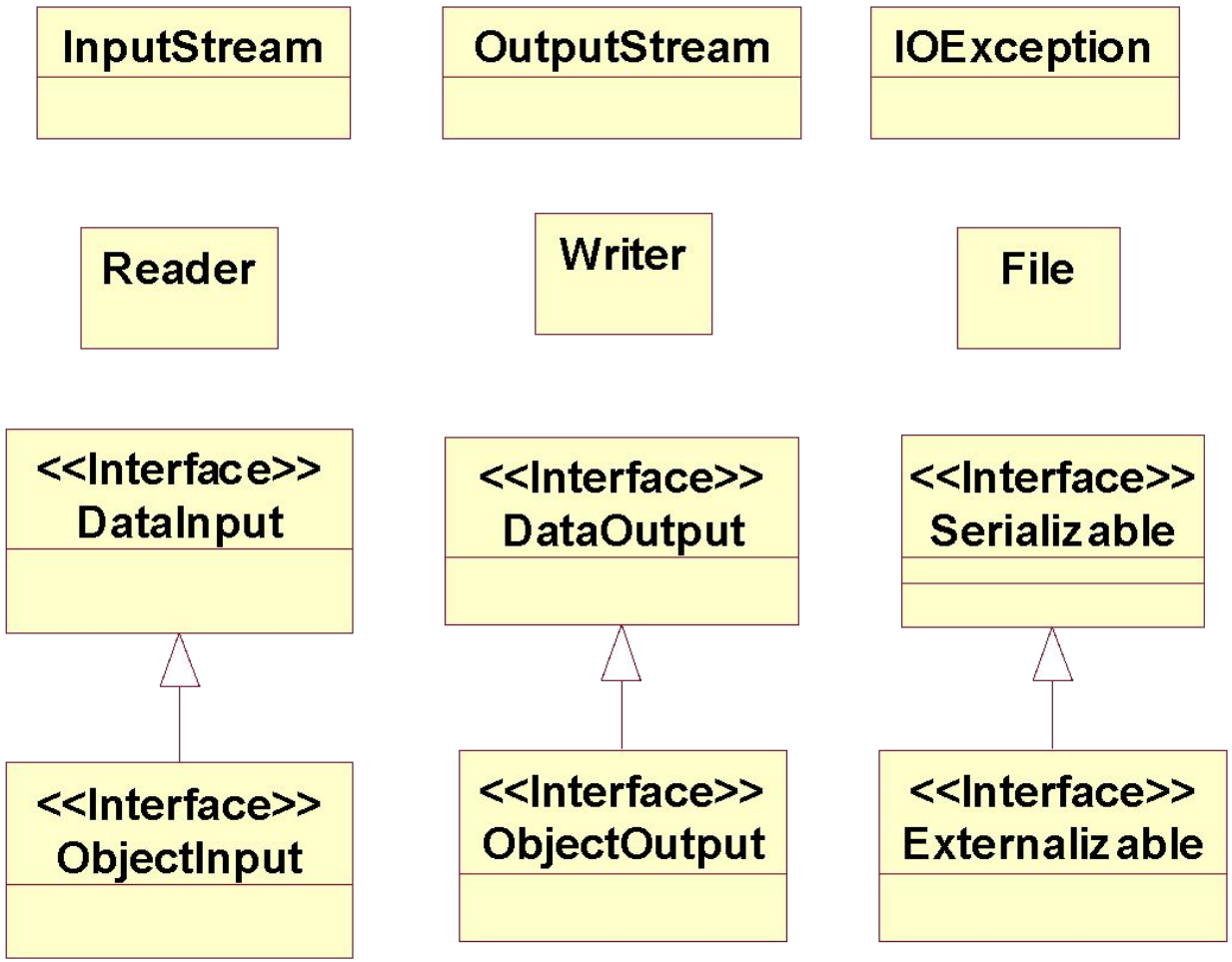
# Таймер и планируемые задания



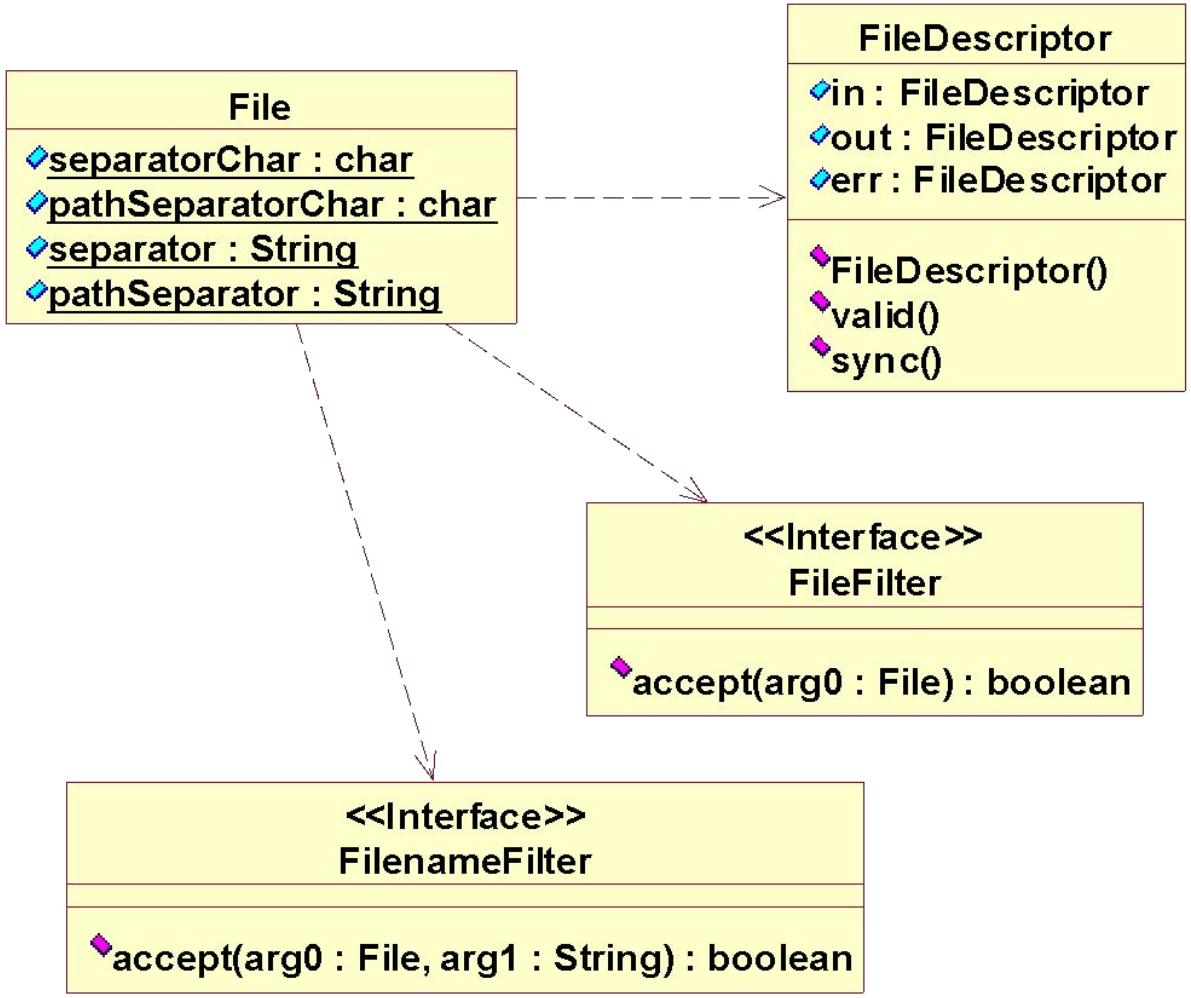
# Интернационализация



# Подсистема ввода-вывода java.io

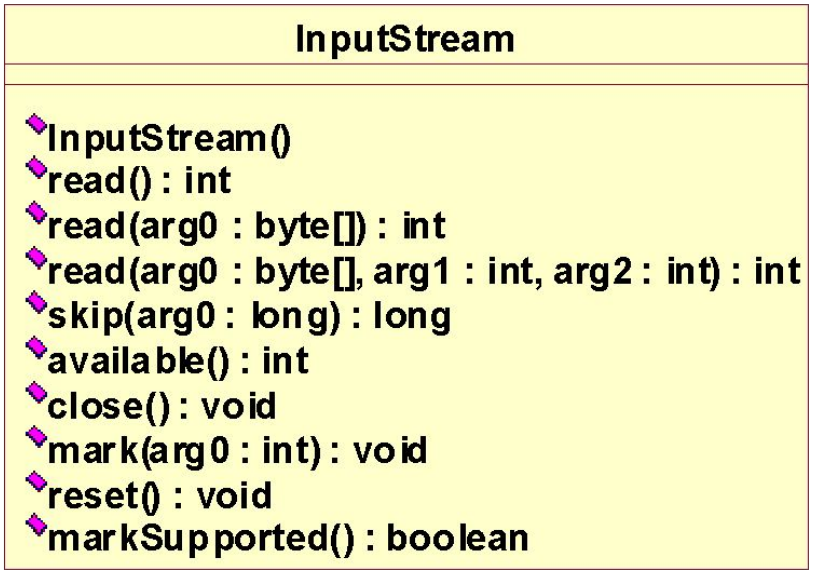


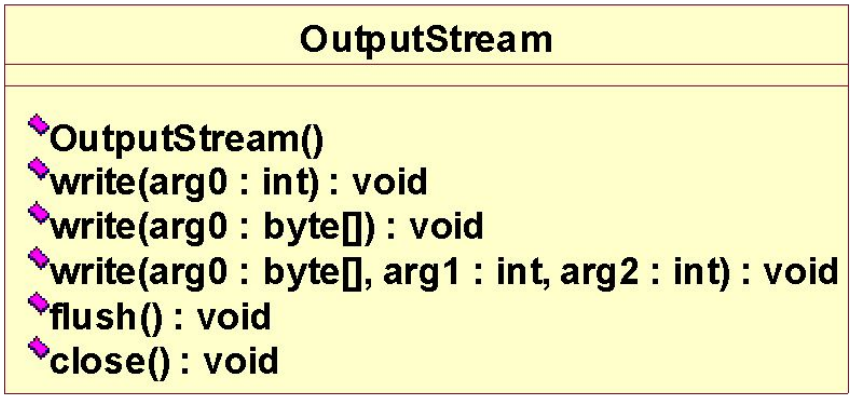
# Средства работы с файлами



# 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



# 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

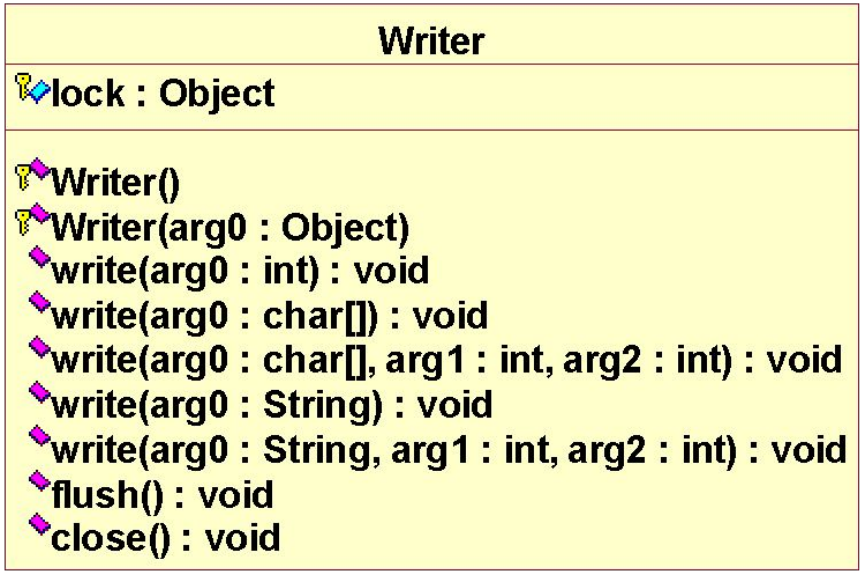
- Фильтрующие потоки предоставляют дополнительную функциональность на основе других потоков
  - абстрактные – FilterInputStream, FilterOutputStream
  - буферизация – BufferedInputStream, BufferedOutputStream
  - форматированный ввод-вывод данных – DataInputStream, DataOutputStream
  - сериализация объектов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

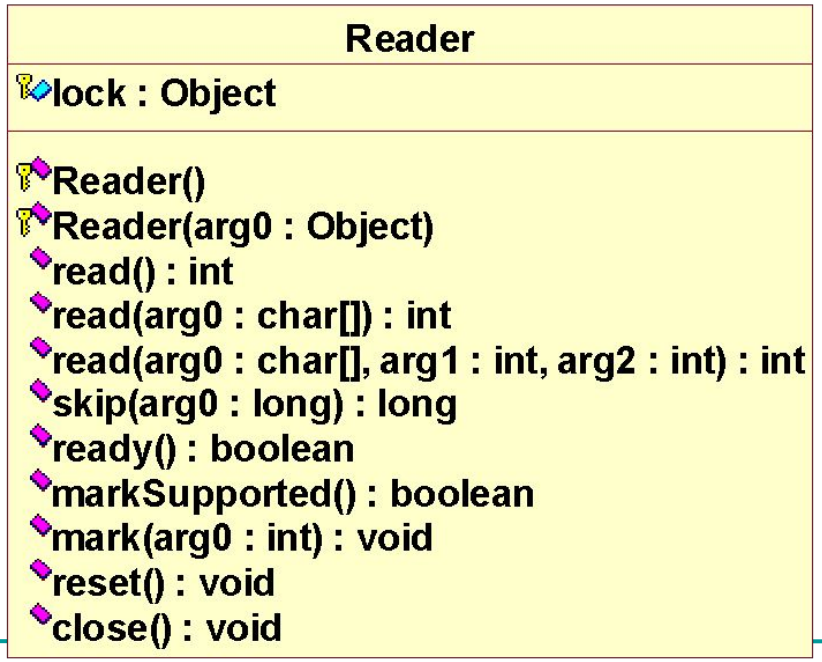
# Произвольный доступ к файлу

- 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

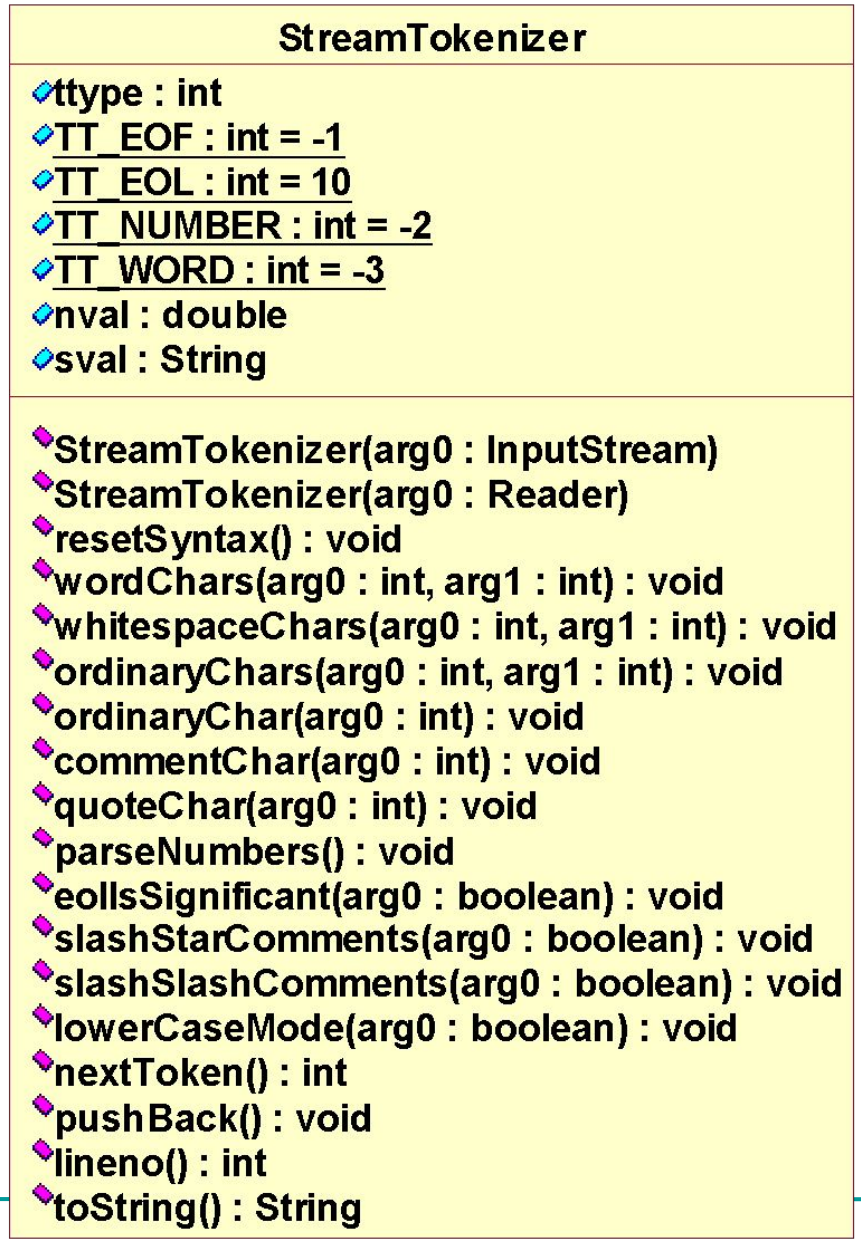


# Символьные потоки

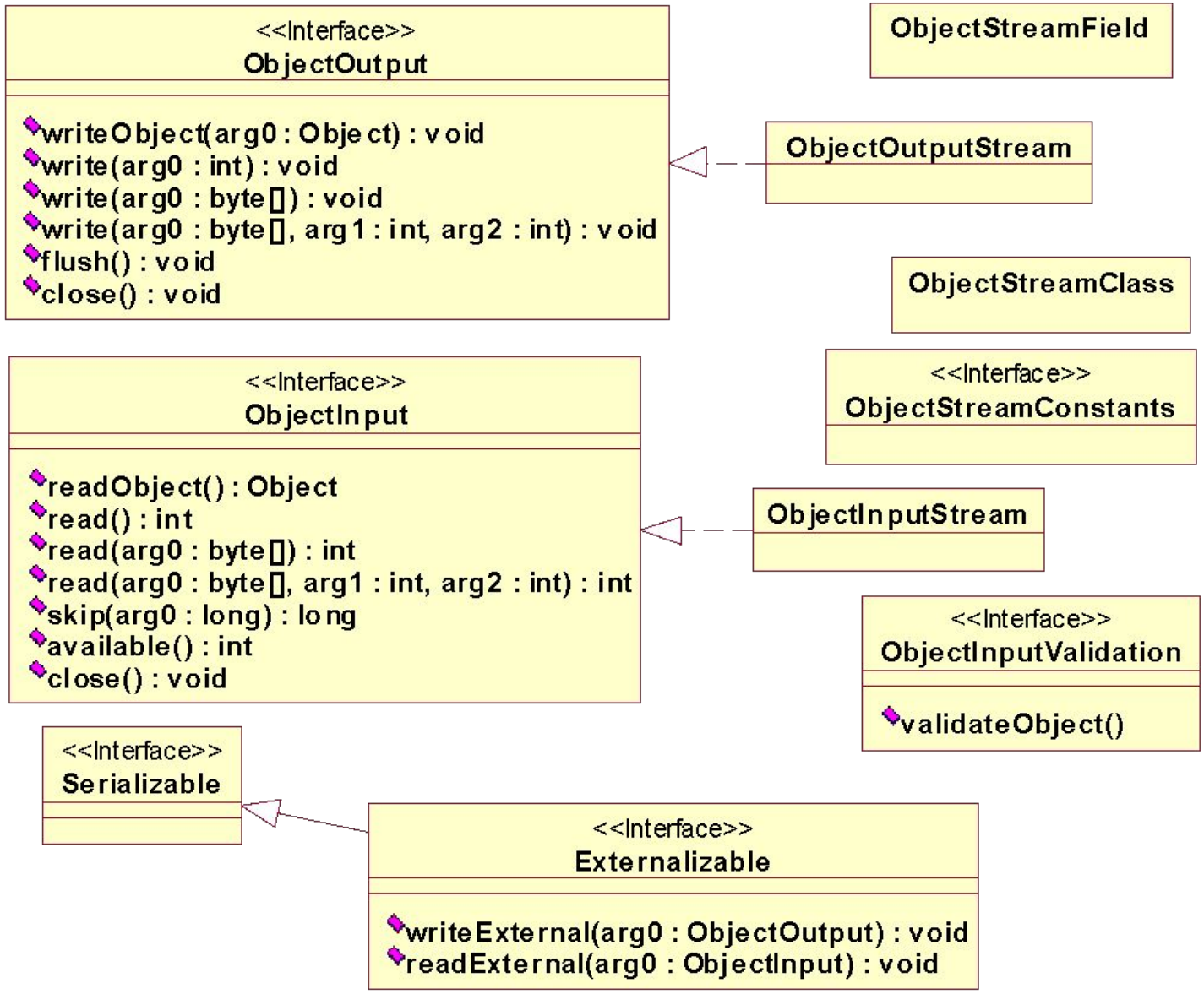
- Узловые символьные потоки:
  - файловые – FileReader, FileWriter
  - на основе массивов символов – CharArrayReader, CharArrayWriter
  - канальные – PipedReader, PipedWriter
  - строковые – StringReader, StringWriter
- Фильтрующие символьные потоки:
  - абстрактные – FilterReader, FilterWriter
  - буферизирующие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

# StreamTokenizer class

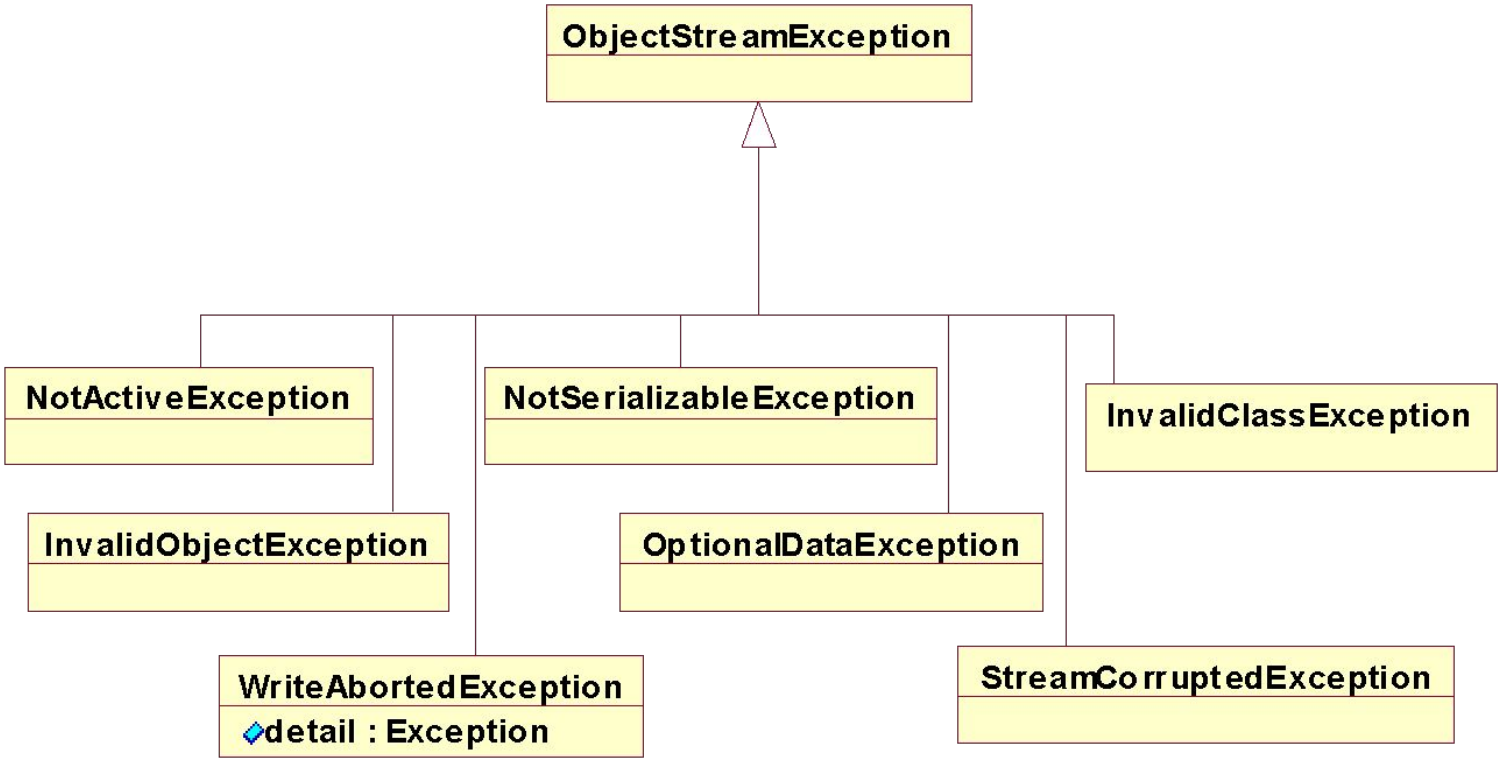


# Сериализация объектов

- 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


<<Interface>>
ObjectOutput
writeObject(arg0 : Object) : void
write(arg0 : int) : void
write(arg0 : byte[]) : void
write(arg0 : byte[], arg1 : int, arg2 : int) : void
flush() : void
close() : void
ObjectStreamField
ObjectOutputStream
ObjectStreamClass
<<Interface>>
ObjectStreamConstants
<<Interface>>
ObjectInput
readObject() : Object
read() : int
read(arg0 : byte[]) : int
read(arg0 : byte[], arg1 : int, arg2 : int) : int
skip(arg0 : long) : long
available() : int
close() : void
ObjectInputStream
<<Interface>>
ObjectInputValidation
validateObject()
<<Interface>>
Serializable
<<Interface>>
Externalizable
writeExternal(arg0 : ObjectOutput) : void
readExternal(arg0 : ObjectInput) : void

# 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 сеариализуются
- 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
- Сериалиазации внутренних или анонимных классов следует категорически избегать

# Управление сериализацией объекта

- Для явного определения списка сериализуемых полей нужно в классе объявить поле private static final ObjectStreamField[] 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 объкта при сериализации/десериализации нужно определить методы:
  - Object writeReplace() throws ObjectStreamException {…}
  - Object readResolve() throws ObjectStreamException {…}
- 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