ИТМО ВТ

3. API. Основные классы

Пакет java.lang

- Не нужно импортировать — неявный import
- классы:
  - Object
  - Number, Integer, Byte, Short, Long, Double, Float, Boolean, Char
  - Math, BigInteger, BigDecimal
  - String, StringBuffer, StringBuilder
  - System, Runtime, ClassLoader, SecurityManager
  - Class, Enum
  - Throwable, Exception, Error, RuntimeException
  - Thread, ThreadGroup
- интерфейсы
  - Cloneable
  - Runnable

Класс Object

- базовый класс иерархии классов Java
- предок всех классов (и массивов)
- неявное расширение

```
class A extends Object {
}
```

Класс Object

- методы класса Object:
  - не переопределяемые
    - `final Class<?> getClass()`
    - `final void wait()`
    - `final void notify()`
    - `final void notifyAll()`
  - переопределяемые
    - `boolean equals()`
    - `int hashCode()`
    - `String toString()`
    - `protected Object clone()`
    - `protected void finalize()`

Метод equals

- boolean equals(Object obj)
- obj1 == obj2 — true, если ссылка на один и тот же объект
- obj1.equals(obj2) — эквивалентность значений
- рефлексивность: x.equals(x) == true
- симметрия: x.equals(y) == y.equals(x)
- транзитивность: x.equals(y) == true && y.equals(z) == true, → x.equals(z) == true
- постоянство
- если x != null, то x.equals(null) == false
- при переопределении — сравнение всех значащих полей

ИТМО ВТ

Метод hashCode()

- int hashCode()
- если объект не модифицировался, то hashCode не должен меняться
- если x.equals(y) == true, то x.hashCode() == y.hashCode()
- если два объекта не эквивалентны, то hashCode не обязан быть разным — коллизия
- по умолчанию обычно возвращает адрес объекта
- при переопределении — скорость, минимизация коллизий

ИТМО ВТ

Метод toString()

- String toString()
- Возвращает текстовое представление объекта
- Неявно вызывается при конкатенации строки и объекта

  "Hello" + obj

  "Hello" + obj.toString()
- Для Object — возвращает ClassName@hashCode$_{16}$

Метод clone()

- protected Object clone()
- Возвращаемый объект следует получать с помощью вызова super.clone()
- Класс должен реализовывать интерфейс Cloneable (интерфейс-метка)
- Object не реализует Cloneable
- Object.clone() создает "мелкую" копию — копируются только значения полей

ИТМО ВТ

Метод finalize()

- protected void finalize()
- Вызывается сборщиком мусора перед удалением объекта
- Предназначен для освобождения системных ресурсов
- Метод Object.finalize() ничего не делает

ИТМО ВТ

Классы оболочки

- Коллекции не могут содержать примитивные элементы
- Оболочка — представление в виде объекта
- abstract class Number
- Byte, Short, Integer, Long, Float, Double: extends Number
- Character
- Boolean

- Void — объектное представление типа void

Number

- Абстрактный класс
- Методы
  - byte byteValue()
  - short shortValue()
  - int intValue()
  - long longValue()
  - float floatValue()
  - double doubleValue()
- Потомки — Byte, Short, Integer, Long, Float, Double

Методы преобразования

- примитив ↔ объект
  - конструктор Integer(int), static Integer valueOf(int)
  - int intValue()
  - автоупаковка/автораспаковка
- строка ↔ объект
  - конструктор Integer(String),static Integer valueOf(String)
  - String toString()
- строка ↔ примитив
  - static int parseInt(String)
  - "" + int

Автоупаковка и автораспаковка

- `Integer a = Integer.valueOf(5);`
- `int i = a.intValue();`

- `Integer a = 5;`
- `int i = a;`

- `Integer a = 500; Integer b = 500;`
  - `b.equals(a) → true; b == a → false`
- `Integer c = 10; Integer d = 10;`
  - `c.equals(d) → true; c == d →` **`true`**!

Методы сравнения

- `interface Comparable<T>`
  - `int compareTo(T)`
- `static` *`prim`* `compare(`*`prim`*`, `*`prim`*`)`
- `boolean equals(Object)`

Integer, Long::работа с битами

- static int bitCount(int)
- static int highestOneBit(int)
- static int lowestOntBit(int)
- static int numberOfLeadingZeros(int)
- static int numberOfTrailingZeros(int)
- static int reverse(int)
- static int rotateLeft(int, int distance)
- static int rotateRight(int, int distance)

Integer, Long :: беззнаковые целые

- compareUnsigned(x, y)
- divideUnsinged(x, y)
- remainderUnsigned(x, y)
- parseUnsignedInt(String) (parseUnsignedLong)
- toUnsignedInt(x) (toUnsignedLong)
- toUnsingedString(x)

Boolean

- compareTo
  - TRUE.compareTo(TRUE) = 0
  - FALSE.compareTo(FALSE) = 0
  - TRUE.compareTo(FALSE) = +
  - FALSE.compareTo(TRUE) = -
- parseBoolean(String), valueOf(String)
  - true, если строка содержит "true" без учета регистра
  - flase, если не содержит

Character

- char — 16 бит
- Unicode (сначала 16 бит, потом 21)
  - U+0000 ... U+10FFFF
  - U+0000 ... U+FFFF — Basic Multilingual Plane
- UTF-16
  - \uD800 ... \uDBFF — high surrogate
  - \uDC00 ... \uDFFF — low surrogate
- int toCodePoint(char high, char low)
- char[] toChars(int codePoint)
- Методы принимают или char или int:
  - isDigit, isLetter, isSpaceChar, isLowerCase, isUpperCase, isTitlecase
  - toLowerCase, toUpperCase, toTitleCase
  - DŽ (upper), Dž (title), dž (lower)

String

- Строка — константа
- String s = "Hello";

  s = s.replace('e', 'u') — создается новый объект
- Компилятор сохраняет строки в пул
- "Hello, " + "world" == "Hello, world"
- метод intern() помещает строку в пул, если ее там нет

Конструкторы String

- String()
- String(byte[])
- String(byte[], int offset, int count)
- String(char[])
- String(char[], int offset, int count)
- String(int[], int offset, int count)
- String(String)
- String(StringBuilder)
- String(StringBuffer)

Методы String

- int length()
- char charAt()
- int codePointAt()
- String concat(String)
- String substring(int begin, int end)
- int indexOf
- int lastIndexOf
- boolean compareTo()
- String replace()
- String trim()
- static String valueOf()

StringBuilder и StringBuffer

- StringBuffer — потокобезопасный
- StringBuilder — быстрый
- Конструкторы
  - StringBuilder()
  - StringBuilder(String)
- Методы
  - String toString()
  - append(x) — используется при операции + со строками
  - insert(int, x)
  - deleteCharAt()
  - setCharAt()
  - replace

Math

- Константы PI, E
- математические функции

System

- System.in, System.out, System.err — стандартные потоки
- Console console()
  - printf(), format()
  - readLine(), readPassword()
- exit(int status)
  - 0 — удачное завершение
- getenv()
- getProperties()
- getSecurityManager()
  - checkXXX()
- currentTimeMillis()
  - nanoTime()

Обработка исключений

- Исключительная ситуация — событие во время исполнения программы, нарушающее нормальный процесс исполнения кода
- 3 типа исключений:
  1) Исключения, порождаемые программным путем
  2) Исключения, возникающие в виртуальной машине
  3) Ошибки
- Исключения могут быть:
  - контролируемые (1) — должны быть обработаны или объявлены в заголовке метода
  - неконтролируемые (2 и 3)

Иерархия классов исключений

- Throwable — аргумент инструкции catch
- потомки: Error и Exception
  - Error — ошибка, обрабатывать бесполезно (OutOfMemory)
  - Exception — контролирумое исключение, должно быть обработано или передано вызывающему методу. Выбрасывается методами в заранее известных участках кода (FileNotFound)
    - подкласс RuntimeException — неконтролируемое исключение, обычно выбрасывается виртуальной машиной. Обычно — баг в логике программы. Может возникнуть в любом месте и в любое время (NullPointer)

Объявление и генерация исключения в методе

```
public void m() throws MyException {
   ...
   throw new MyException("No data file");
   ...
}

public class MyException extends Exception {
}
```

Обработка исключения

- Правило "Catch or Specify"

```
try {
    m();
} catch (MyException e) {
    System.out.println("A-a-a! Мы все умрём!");
    System.exit(666);
} catch (Exception e) {
    System.out.println("Или не все...");
} finally {
    // закрыть ресурсы
}
```

finally выполняется всегда!

```
public int test() {
  try {
    return 0;
  } finally {
    return 1;
  }
}

System.out.println(test());
```

try с ресурсами

- ресурс должен реализовывать интерфейс AutoCloseable

```
try (Scanner s = new Scanner(System.in)) {
   ...
} catch (IOException e) {
}
```

- при выходе из блока try будет вызван метод close()

Потоки выполнения

- Процесс имеет собственный контекст исполнения, собственный набор ресурсов, свою выделенную память.
- Поток (thread) существует внутри процесса, делит память и ресурсы с другими потоками.
- Потоки в JVM
  - системные потоки:
    - основной поток виртуальной машины
    - сборщик мусора
    - поток периодических задач
    - поток динамической компиляции
  - прикладные потоки
    - основной поток (main)
    - создаваемые программным путем

Класс Thread и интерфейс Runnable

- public class Thread implements Runnable
- 2 варианта создания потока

1)
```
public class A extends Thread {
    public void run() { /* тело потока*/ }
}
new A().start;
```

2)
```
public class A implements Runnable {
    public void run() { /* тело потока */ }
}
new Thread(new A()).start();
```

Приостановка и прерывание потока

- try {
  Thread.sleep(1000); // спать 1 с
  } catch (InterruptedException e) { }
- try {
  t.join(1000); // ожидать завершения t в течение 1 с
  } catch (InterruptedException e) { }
- t.interrupt(); // устанавливает флаг прерывания
- if (Thread.interrupted() { // сбрасывает флаг
  throw new InterruptedException();
  }

Множественный доступ к переменным

```
class A {
  int counter = 0;
  public void up() { counter++; }
  public void down() { counter--; }
}
```

1) получить значение counter

2) увеличить/уменьшить значение на 1

3) сохранить новое значение

действие	**1)**	**1)**	**2)**	**2)**	**3)**	**3)**
стек	**0**	**0**	**-1**	**1**	**1**	**-1**
counter	**0**	**0**	**0**	**0**	**1**	**-1**

Синхронизация

```
class A {
  int counter = 0;
  public synchronized void up() { counter++; }
  public synchronized void down() { counter--; }
}
```

- Любой объект имеет монитор
- Поток может выполнить synchronized метод, только захватив монитор
- При выходе из метода поток освобождает монитор
- Для уменьшения времени захвата монитора используются блоки synchronized

```
synchronized(Object) { }
```

Модификатор volatile

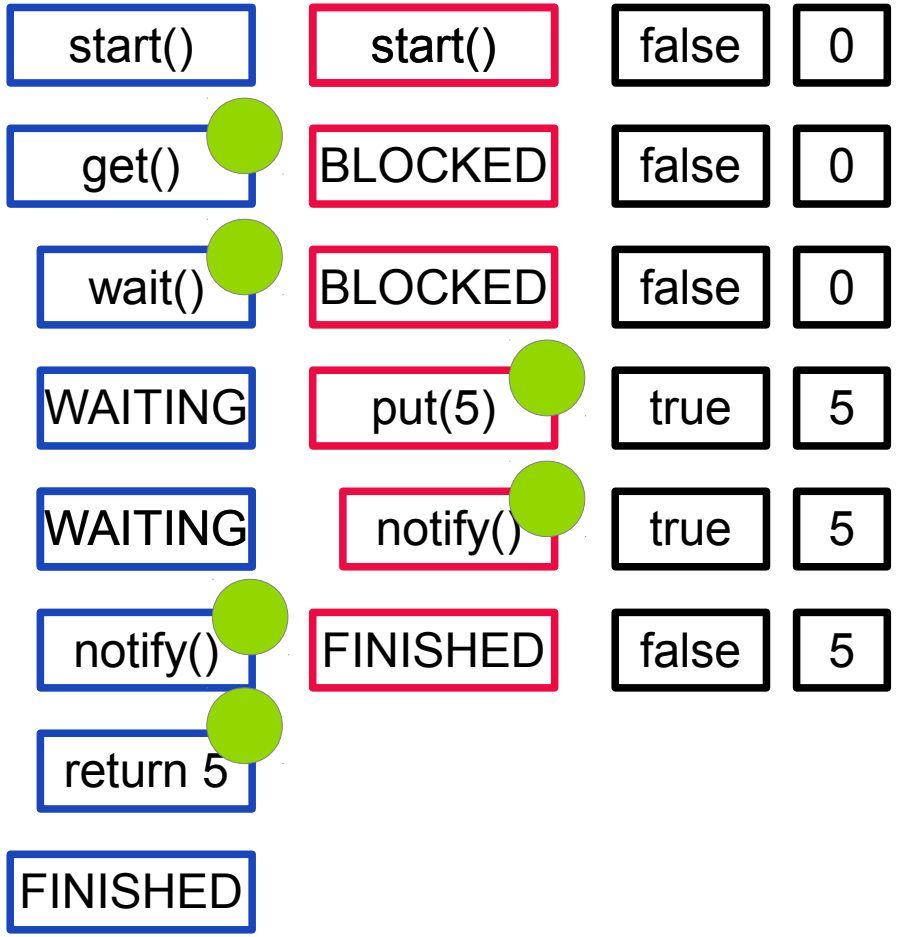
- Переменные класса — общие для всех потоков
- Потоки могут сохранять значения общих переменных в локальном кэше
- Модификатор volatile указывает, что доступ к этой переменной из потоков должен производиться напрямую

Взаимодействие потоков

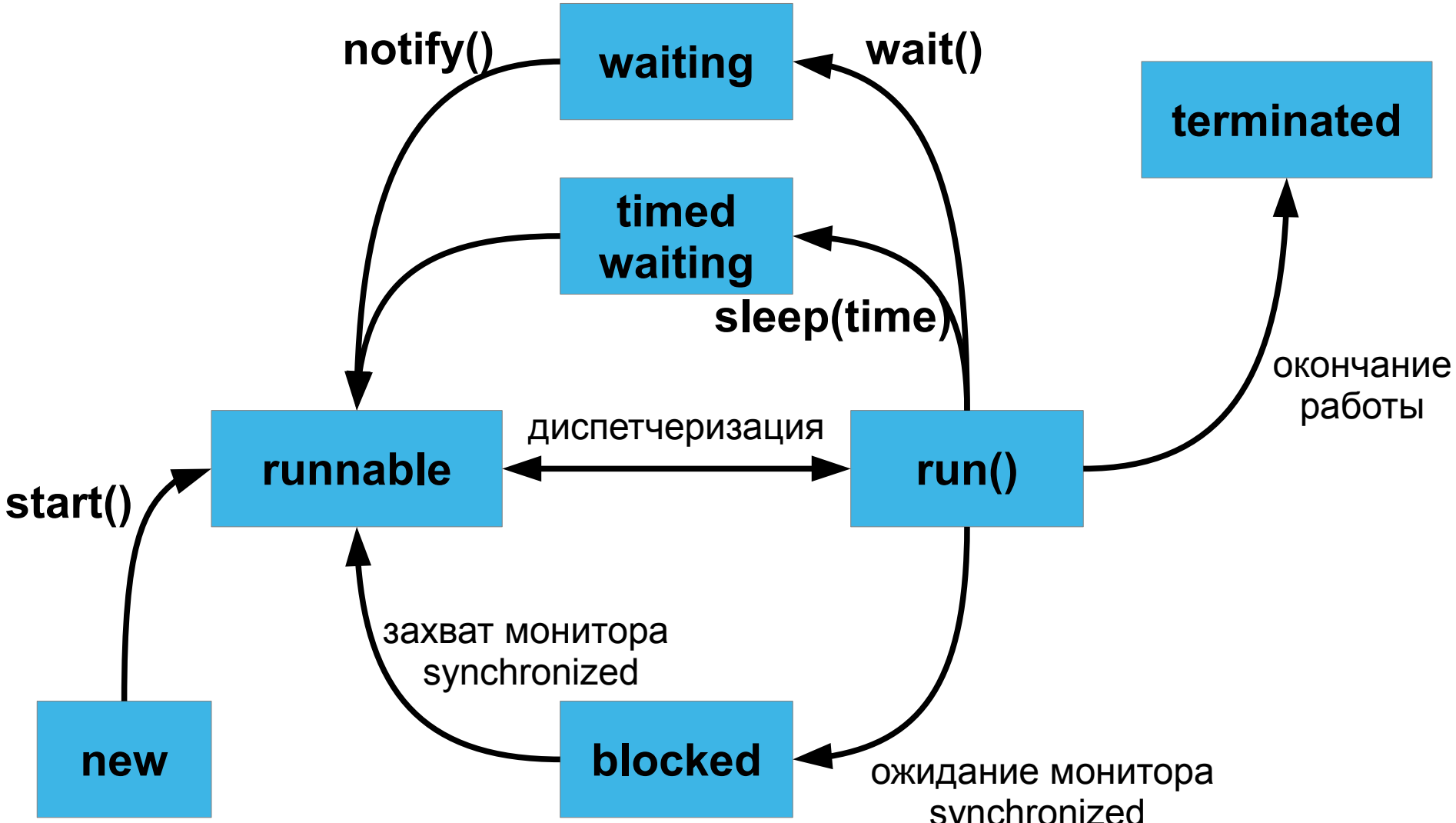
- Методы wait(), notify(), notifyAll()
- Для вызова этих методов поток должен иметь монитор данного объекта, поэтому они вызываются в синхронизированном блоке или методе.
- wait() - поток помещается в список ожидания и освобождает монитор. После выхода из списка ожидания, он может получить монитор, и завершить метод wait.
- notify() - выводит из списка ожидания один из потоков.
- notifyAll() - выводит из списка ожидания все потоки.

Диаграмма работы wait/notify

```
class A {
  boolean flag = false;
  int value;
  synchronized void put(int i) {
    while(flag) { wait(); }
    flag = true; value = i;
    notifyAll();
  }
  synchronized int get() {
    while(!flag) { wait(); }
    flag = false;
    notifyAll();
    return value;
  }
}
```



Состояния потока



Проблемы многопоточности

- Взаимная блокировка (deadlock)
- Ресурсное голодание (starvation)
- Зацикливание (livelock)

пакет java.util.concurrent

- Исполнители
  - interface Executor : void execute(Runnable)
  - interface ExecutorService : Future submit(Callable)
  - interface Callable<V> : V call()
  - interface Future<V> : V get()
  - interface ScheduledExecutorService : ScheduledFuture schedule()
- Реализации
  - class ThreadPoolExecutor
  - class ScheduledThreadPoolExecutor
- Вспомогательный класс
  - Executors
    - создание ExecutorService, ScheduledExecutorService, Callable

пакет java.util.concurrent.locks

- интерфейс Lock

```
Lock l = new ReentrantLock();
l.lock(); // tryLock()
try { /* доступ к защищенному ресурсу */ }
finally { l.unlock(); }
```

- интерфейс Condition

```
Condition c = l.newCondition();
l.lock();
try {
  while(flag) { c.await(); }
  flag = true;
  c.signal();
} finally { l.unlock(); }
```

пакет java.util.concurrent.atomic

- Содержит классы для реализации атомарных операций

```
class A {
  AtomicInteger counter = new AtomicInteger(0);
  public void up() { counter.incrementAndGet(); }
  public void down() { counter.decrementAndGet(); }
}
```

java.lang.reflect

- Класс java.lang.Class
- Получение объекта класса Class
  - new A().getClass();
  - Class.forName("A");
  - A.class
- Методы
  - Field[] getFields()
  - Method[] getMethods()
  - Constructor[] getConstructors()
  - isInterface()
  - isArray()

4. Коллекции

Коллекции

- Коллекции предназначены для группировки объектов
- Коллекции могут хранить только ссылочные типы
- Коллекции используют обобщенные типы данных

Обобщенные типы

- ```
  Stack a = new Stack();
  a.put("Hello"); // void put(Object)
  String s = (String) a.get(); // Object get()
  ```
- ```
  Stack<String> b = new Stack<String>(); // Stack<>()
  b.put("Hello");
  String s = b.get();
  ```
- ```
  class Stack<T> {
    private T object;
    void put(T t) { object = t; }
    T get() { return object; }
  }
  ```
  ```

Ограничение параметров типа

```
class NumberStack<T extends Number> {
  public int intValue() {
    return T.intValue();
  }
}
```

Приведение типов и шаблоны

- `Number n = new Integer(10);`
- `Stack<Number> s = new Stack<Integer>(); // error`

  `Stack<Integer> не потомок Stack<Number>`
- `Stack<?> s = new Stack<Integer>();`
- `Stack<? extends Integer> = new Stack<Integer>();`

Стирание типов

- На уровне виртуальной машины обобщенные типы отсутствуют — заменяются на "сырые"
- Добавляется приведение типов
- Совместимость с предыдущими версиями
- Потенциальная возможность несовпадения типов

Интерфейс Collection<E>

- int size()
- boolean isEmpty()
- boolean contains(Object)
- booelan add(E) // true если коллекция изменилась
- boolean remove(Object) // true если коллекция изменилась
- Iterator<E> iterator()
- void clear()
- containsAll(), addAll(), removeAll(), retainAll()
- for (E e : collection) { e }
- for (Iterator i = collection.iterator(); i.hasNext();) { i.next(); }

Интерфейс Set<E>

- Множество (нет повторяющихся элементов)
- s.containsAll(t) - true, если t — подмножество s
- s.addAll(t) — объединение s и t
- s.retainAll(t) — пересечение s и t
- s.removeAll(t) — разность s и t

- HashSet — самая быстрая реализация, нет порядка
- TreeSet — медленная реализация, натуральный порядок
- LinkedHashSet — средняя скорость, порядок добавления

- Set<Integer> s = new HashSet<Integer>();

Интерфейс List<E>

- Список или последовательность — индексация
- E get(int)
- E set(int, E)
- void add(int, E)
- E remove(int)
- indexOf(Object)
- lastIndexOf(Object)
- listIterator() // hasPrevious(), previous()
- ArrayList — оптимальная реализация
- LinkedList — для большого числа вставок-удалений

Интерфейс Queue<E>

- Очередь
- add(E), E remove(), E element() - исключение
- offer(E), E poll(), E peek() - специальное значение
- LinkedList — сортировка FIFO
- PriorotyQueue — сортировка по значению

Интерфейс Deque<E>

- Очередь + стек
- addFirst/Last(E), E removeFist/Last(), E getFirst/Last()
- offerFirst/Last(E), E pollFirst/Last(), E peekFirst/Last()
- ArrayDeque
- LinkedList

Интерфейс Map<K,V>

- Хранит пары ключ-значение
- V put(K,V)
- V get(K)
- V remove(K)
- boolean containsKey(K)
- boolean containsValue(V)
- int size()
- boolean isEmpty()
- Set keySet()
- Set entrySet()
- Collection values()

- HashMap
- TreeMap
- LinkedHashMap

класс Collections

- статические методы, возвращающие:
  - синхронизированные коллекции synchronizedCollection()
  - немодифицируемые коллекции unmodifiableCollection()
  - проверяемые коллекции checkedCollection()
- алгоритмы
  - sort()
  - shuffle()
  - reverse()
  - fill()
  - swap()
  - binarySearch()

Сортировка (отдельный компаратор)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(new C());
    for (String s : list) { System.out.println(s); }
    }
}
class C implements Comparator<String> {
  public compare(String s1, String s2) {
    return s1.length() - s2.length();
  }
}
```

Сортировка (статический вложенный класс)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(new C());
    for (String s : list) { System.out.println(s); }
  }
  static class C implements Comparator<String> {
    public compare(String s1, String s2) {
      return s1.length - s2.length;
    }
  }
}
```

Сортировка (внутренний класс)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(new A().new C());
    for (String s : list) { System.out.println(s); }
  }
  class C implements Comparator<String> {
    public compare(String s1, String s2) {
      return s1.length() - s2.length();
    }
  }
}
```

Сортировка (локальный класс)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    class C implements Comparator<String> {
      public compare(String s1, String s2) {
        return s1.length() - s2.length();
      }
    }
    list.sort(new C());
    for (String s : list) { System.out.println(s); }
  }
}
```

Сортировка (анонимный класс)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(new Comparator<String>() {
      public compare(String s1, String s2) {
        return s1.length() - s2.length();
      }
    });
    for (String s : list) { System.out.println(s); }
  }
}
```

Сортировка (лямбда-выражение)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(
      (String s1, String s2) -> s1.length() - s2.length()
    );
    for (String s : list) { System.out.println(s); }
  }
}
```

- sort(Comparator c)
- @FunctionalInterface interface Comparator
- int compare(T o1, T o2);

Сортировка и вывод списка (лямбда)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(
      (String s1, String s2) -> s1.length() - s2.length()
    );
    list.forEach((s) -> System.out.println(s));
  }
}
```

Сортировка и вывод списка (ссылка на метод)

```
public class A {
  public static void main(String[] args) {
    List<String> list = Arrays.asList(args);
    list.sort(
      (String s1, String s2) -> s1.length() - s2.length()
    );
    list.forEach(System.out::println);
  }
}
```