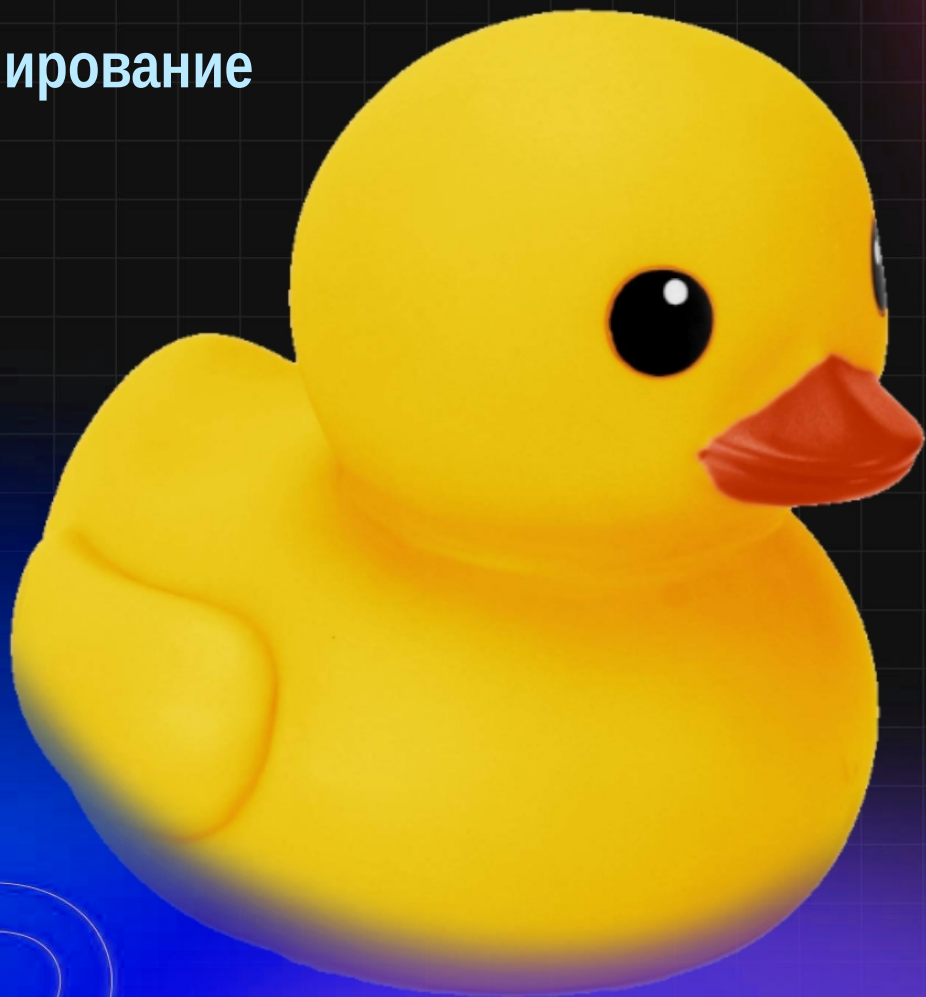


Программирование
2 семестр
2024



ІТМО

Введение

- Шаблоны проектирования
- Дженерики и Коллекции
- Ввод-вывод
- Функциональное программирование
- Сетевое взаимодействие
- Многопоточность
- Работа с базами данных
- Графический интерфейс и локализация

● Гугложурнал

- ❖ лаба 5 (60%...100%)
- ❖ лабы 6 и 7 (60%...100%)
- ❖ лаба 8 (60%...100%)
- ❖ рубеж (60%...100%)
= допуск
- ❖ экзамен (60%...100%)
- ❖ + 3 ЛК

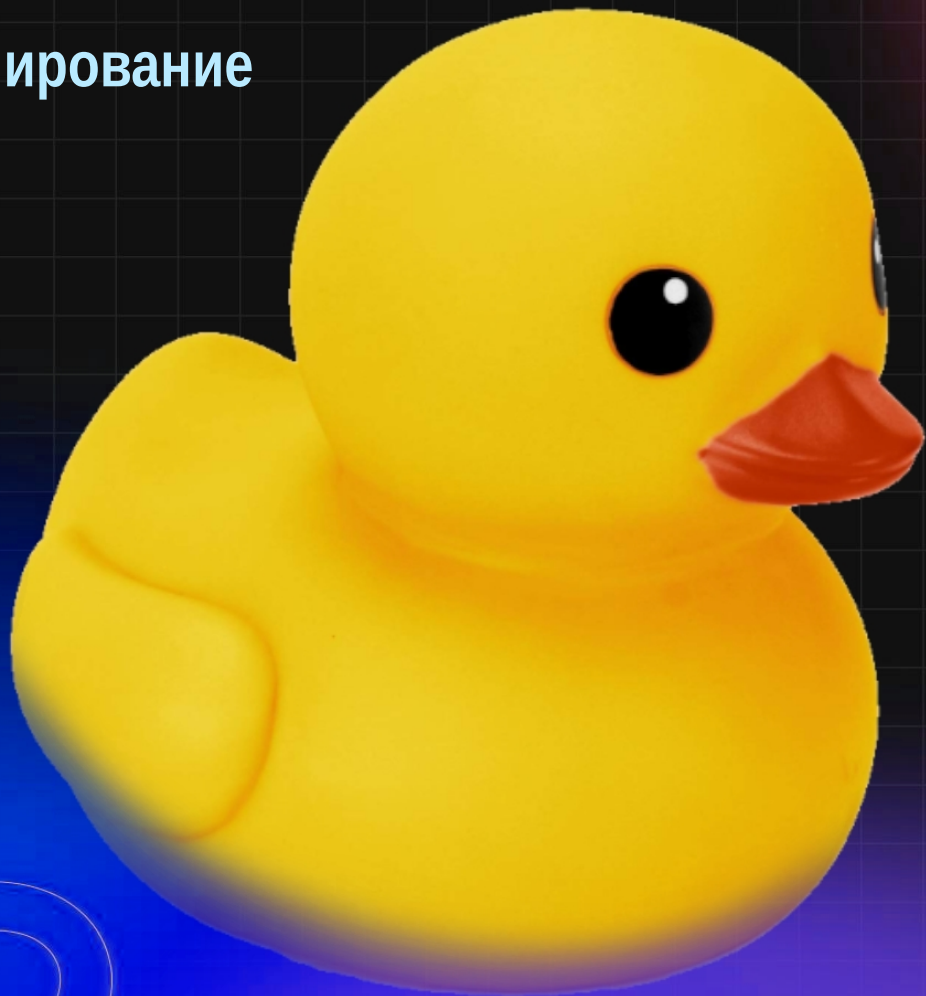
● БаРС

- ❖ лаба 1 (12...20)
- ❖ лаба 2 (24...40)
нужна для 4 или 5
- ❖ рубеж (12...20)
автомат: 3Е: ≥ 60 5А: > 70
- ❖ экзамен (12...20)
- ❖ + 3 ЛК

- <https://se.ifmo.ru>
 - ❖ задания к лабораторным
 - ❖ видео и тексты лекций
 - ❖ методички
- <https://docs.oracle.com/javase/8/docs/api/>
- <https://docs.oracle.com/en/java/javase/17/docs/api/>
- <https://docs.oracle.com/en/java/javase/21/docs/api/>



Программирование
2 семестр
2024



ІТМО

Типы данных
Обертки
(Wrappers)

- Тип данных
 - ❖ диапазон значений + операции
- Типобезопасность
 - ❖ Проверка совместимости типа
- Статическая и динамическая типизация
 - ❖ момент связывания переменной с типом
- Сильная и слабая типизация
 - ❖ явное и неявное приведение



- Статическая типизация
 - ❖ тип переменной при объявлении
 - ❖ компилятор контролирует типы
 - ❖ оптимизация на уровне машинных кодов
- Динамическая типизация
 - ❖ тип есть у значения, у переменной - при присваивании
 - ❖ код проще и более гибкий
 - ❖ больше ошибок

- Сильная
 - ❖ минимум неявного приведения
 - ❖ запрет операций над несовместимыми типами
- Слабая
 - ❖ неявное приведение
 - ❖ операции с разными типами
 - ❖ JavaScript
 - WAT - Gary Bernhardt, 2012
 - <https://www.destroyallsoftware.com/talks/wat>



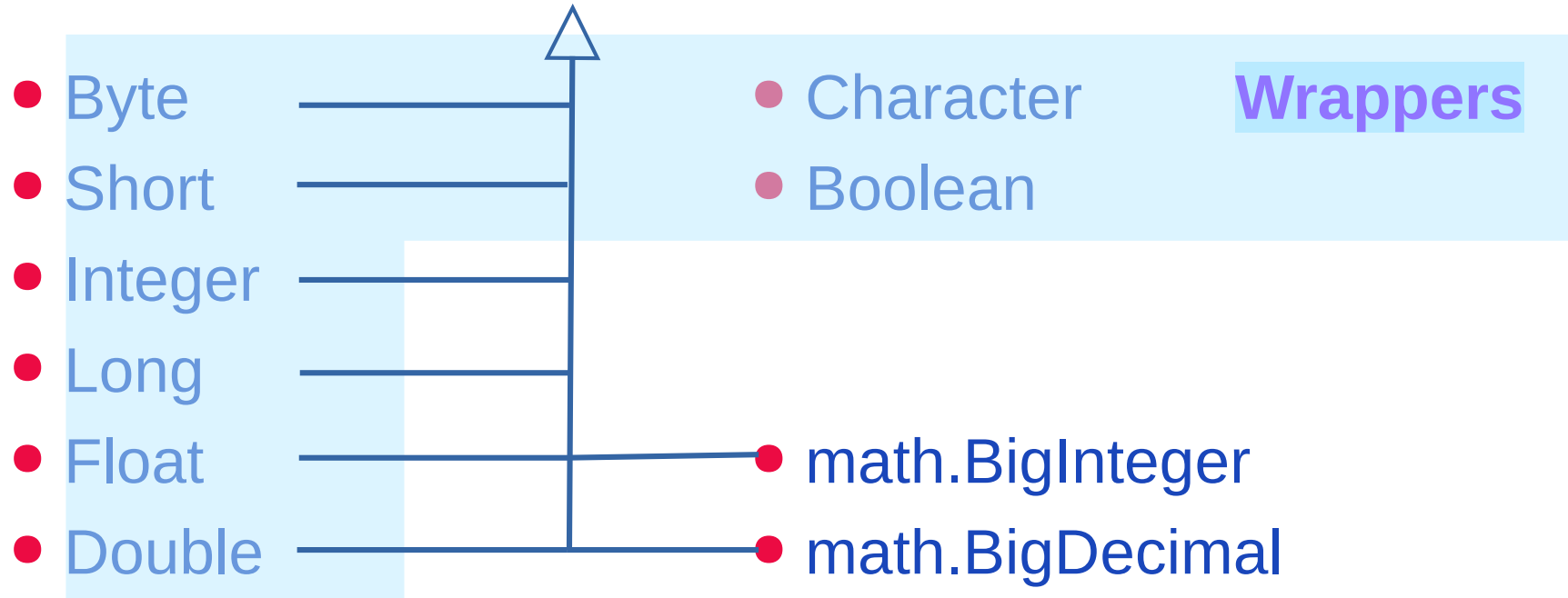
- Примитивные

- ❖ значение
- ❖ не наследник Object
- ❖ эффективные
- ❖ операции
- ❖ массивы
- ❖ не обобщается

- Ссылочные

- ❖ ссылка
- ❖ наследник Object
- ❖ больше и медленнее
- ❖ методы
- ❖ массивы и коллекции
- ❖ обобщается

- Number



```
// int -> Integer
Integer x = Integer.valueOf(9111);
Integer x = 9111; // автоупаковка

// Integer -> int
int y = x.intValue();
int y = x; // автораспаковка

y += 888; // так можно
for (i = 0; i < 10000000000; i++) { // а так не надо
    x += 888; // x = Integer.valueOf(x.intValue()+888);
```



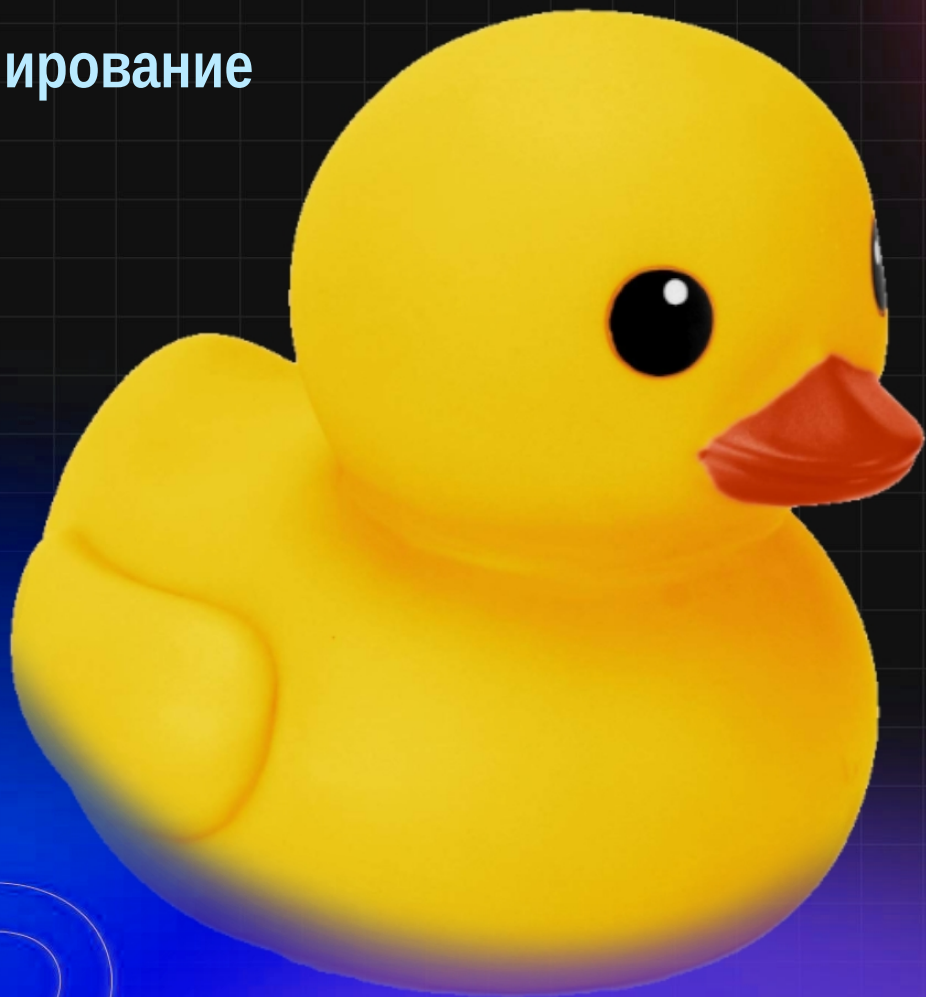
```
Integer b = 9999;  
Integer x = 100;
```

```
b > x      // true (автораспаковка)
```

```
b = 100;  
b.equals(x) // true  
b == x      // true !!! если -128 <= b <= _127
```

- Стандарт Unicode (было 16 бит, стало - 21)
- char (16 бит) U+0000 ... U+FFFF - UTF-16 code unit
 - ❖ U+D800 ... U+DBFF - high surrogate
 - ❖ U+DC00 ... U+DFFF - low surrogate
- int (21 бит) U+0000 ... U+10FFFF - code point
- Методы:
 - ❖ Character valueOf(char), char charValue()
 - ❖ chars[] toChars(int), int toCodePoint(char high, char low)

Программирование
2 семестр
2024



ІТМО

Обобщения
Generics

- Обобщенное программирование
 - ❖ абстрактное описание данных и алгоритмов, применяемое без изменения к разным типам данных
- Параметрический полиморфизм
 - ❖ тип данных является параметром
- Дополнительный контроль на этапе компиляции
 - ❖ более безопасный код

- Перегрузка методов (overriding)
 - ❖ разные типы, разный код,
 - ❖ решение на этапе компиляции
- Полиморфизм подтипов (ООП, overloading)
 - ❖ типы-наследники, перегруженный код,
 - ❖ решение на этапе выполнения
- Параметрический полиморфизм
 - ❖ разные типы, один код,
 - ❖ тип является параметром


```
public class Box {  
    private Object obj;  
  
    public Box(Object o) {  
        obj = o;  
    }  
  
    public Object get() {  
        return obj;  
    }  
}
```

```
var x = "Hello";  
Box box = new Box(x);  
var y = box.get();
```



```
public class Box {  
    private Object obj;  
  
    public Box(Object o) {  
        obj = o;  
    }  
  
    public Object get() {  
        return obj;  
    }  
}
```

```
var x = "Hello";  
Box box = new Box(x);  
var y = box.get();
```

```
int len = y.length();
```

```
error: cannot find symbol  
symbol:   method length()  
location: class Object
```

```
public class Box {  
    private Object obj;  
  
    public Box(Object o) {  
        obj = o;  
    }  
  
    public Object get() {  
        return obj;  
    }  
}
```

```
var x = "Hello";  
Box box = new Box(x);  
String y = box.get();  
  
int len = y.length();  
  
error: incompatible types: Object  
cannot be converted to String  
        String y = box.get();
```

```
public class Box {  
    private Object obj;  
  
    public Box(Object o) {  
        obj = o;  
    }  
  
    public Object get() {  
        return obj;  
    }  
}
```

```
var x = "Hello";  
Box box = new Box(x);  
String y = (String) box.get();  
  
int len = y.length();  
  
// компіляція без помилок!
```

```
public class Box {  
    private Object obj;  
  
    public Box(Object o) {  
        obj = o;  
    }  
  
    public Object get() {  
        return obj;  
    }  
}
```

```
var x = 256;  
Box box = new Box(x);  
String y = (String) box.get();  
  
int len = y.length();  
  
// компиляция без ошибок!  
  
Exception in thread "main"  
java.lang.ClassCastException:  
class java.lang.Integer cannot be cast  
to class java.lang.String  
  
// ошибка при выполнении!!
```

Пример с обобщениями

```
public class Box {  
    private Object obj;  
  
    public Box(Object o) {  
        obj = o;  
    }  
  
    public Object get() {  
        return obj;  
    }  
}
```

```
public class Box<T> {  
    private T obj;  
  
    public Box(T o) {  
        obj = o;  
    }  
  
    public T get() {  
        return obj;  
    }  
}
```

```
var x = "Hello";  
Box<String> box = new Box(x);  
String y = box.get();  
  
int len = y.length();  
  
// компиляция без ошибок!  
  
// выполнение без ошибок!
```

```
public class Box<T> {  
    private T obj;  
  
    public Box(T o) {  
        obj = o;  
    }  
  
    public T get() {  
        return obj;  
    }  
}
```

```
var x = 256;  
Box<String> box = new Box(x);  
String y = box.get();
```

```
int len = y.length();
```

```
// ошибка при компиляции!
```

```
error: incompatible types: Integer  
cannot be converted to String
```

```
public class Box<T> {  
    private T obj;  
  
    public Box(T o) {  
        obj = o;  
    }  
  
    public T get() {  
        return obj;  
    }  
}
```



```
public class Box<T> {
```

Обобщенный тип
(generic)

```
Box<String> box = new Box(x)
```

Параметризованный тип
(parametrized)

- E — элемент коллекции (в Java API)
- K — ключ
- V — значение
- N — число
- T — первый параметр типа
- S, U, V — второй, третий, четвертый параметр типа

- C++ - templates
 - ❖ **разный код** для каждого параметра
 - ❖ во время исполнения **недоступна** информация о шаблонах
- C# - generics
 - ❖ **разный код** для каждого примитивного параметра
 - ❖ во время исполнения **доступна** информация об обобщениях
- Java - generics
 - ❖ при компиляции создается **код без обобщений (type erasure)**
 - ❖ во время исполнения **недоступна** информация об обобщениях

- В байткоде все обобщения стираются - type erasure
- Остаются только базовые типы - raw types
- Компилятор создает мостовые методы - bridge methods

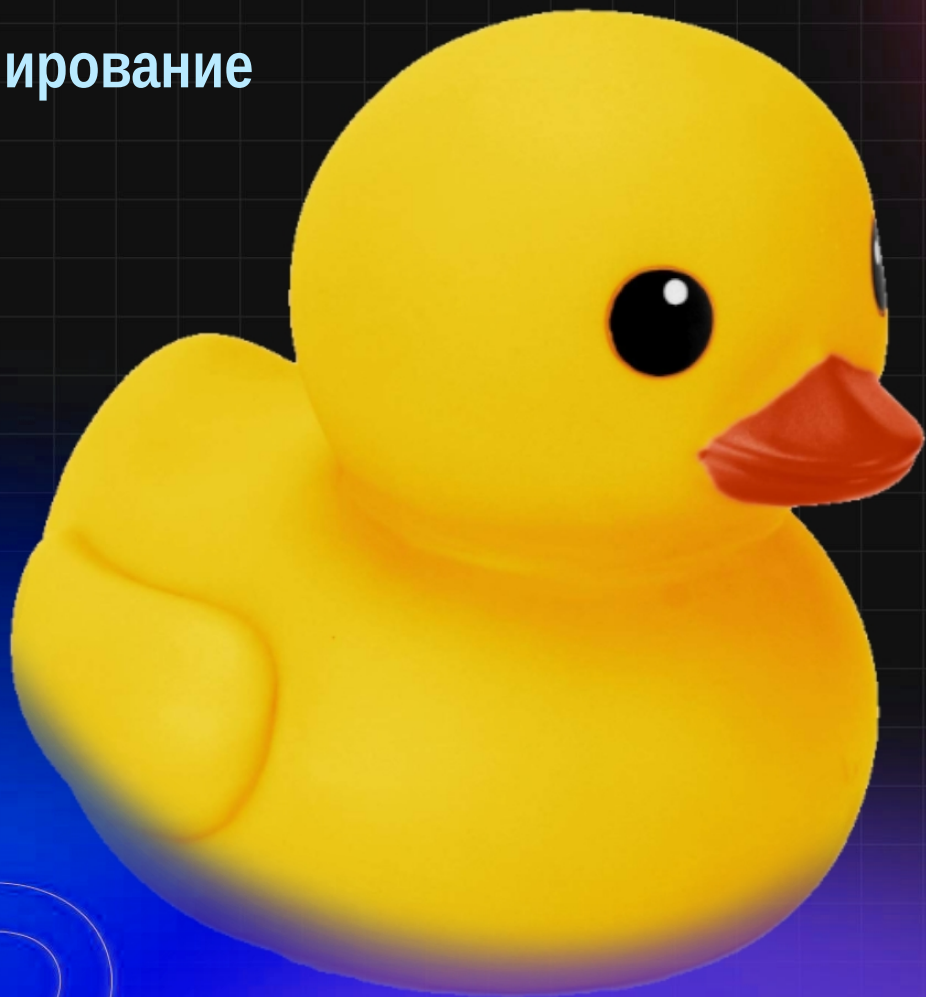
```
class Box<T> {  
    void put(T x) { ... }  
}  
class BBox extends Box<Byte> {  
    void put(Byte b) {  
        super.put(b);  
    }  
}
```

```
class Box {  
    void put(Object x) { ... }  
}  
class BBox extends Box {  
    void put(Byte b) {  
        super.put(b);  
    }  
    void put(Object x) {  
        put((Byte)x);  
    }  
}
```

bridge method

- Совместимость со старыми версиями
- Недостаточная типобезопасность
 - ❖ Контроль только при компиляции
 - ❖ При выполнении возможны нарушения
- Нельзя:
 - ❖ Использовать generic примитив
 - `Box<int>`
 - ❖ Создавать объект параметра типа
 - `E e = new E()`
 - ❖ Создавать массивы обобщений
 - `Box<Cat>[]`
 - ❖ Перегружать методы с обобщениями
 - `void print(Box<Integer>)`
 - `void print(Box<String>)`

Программирование
2 семестр
2024



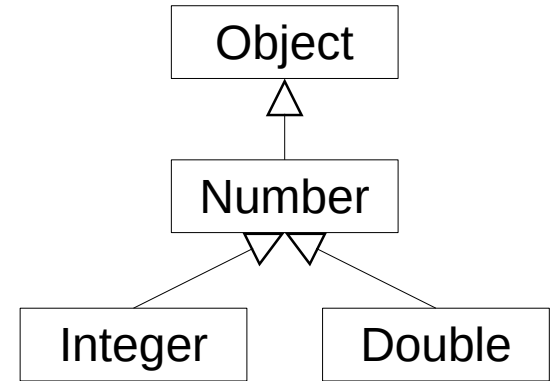
ІТМО

Обобщенные
методы

- Параметр типа перед типом возвращаемого значения
`public static <T> Box<T> pack(T obj)`
- При вызове метода нужен явный или неявный параметр
`Box<Integer> b1 = Box.<Integer>pack(1);`
`Box<String> b2 = Box.pack("Hello");`

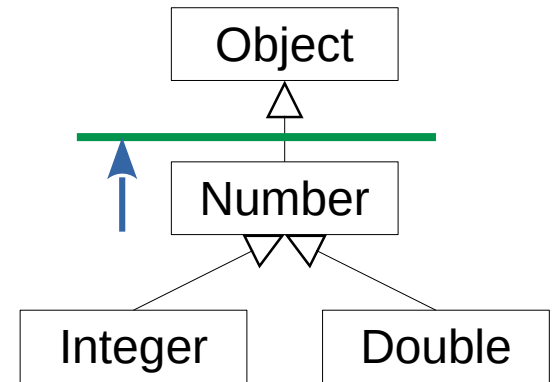
- `Box<T>`, `T` - любой тип (потомок `Object`)

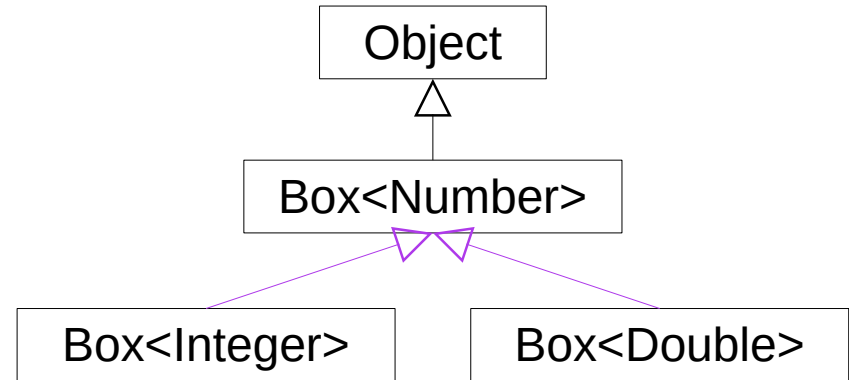
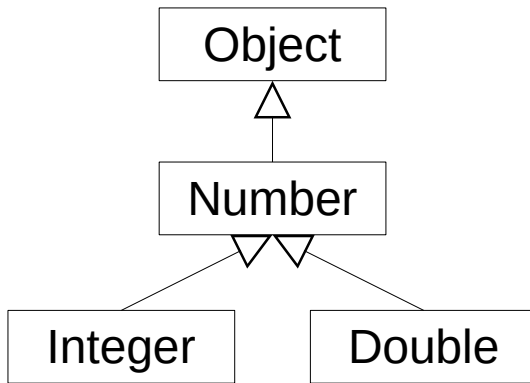
```
public class NumBox<T> {  
    private Object obj  
  
    public int incAndGet() {  
        obj += 1;  
        return obj;  
    }  
}
```



- Инкремент только для чисел

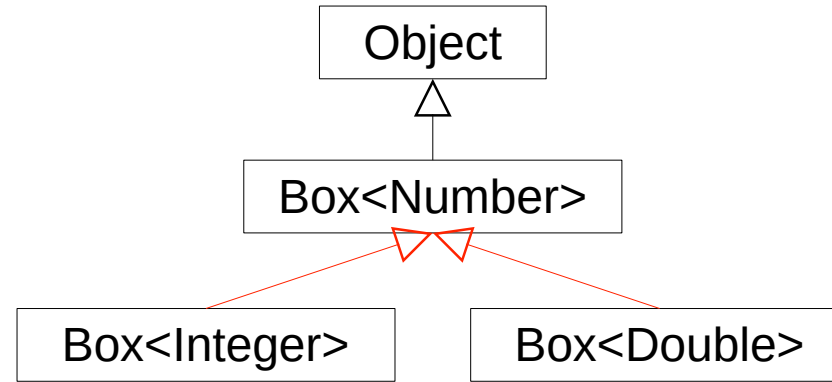
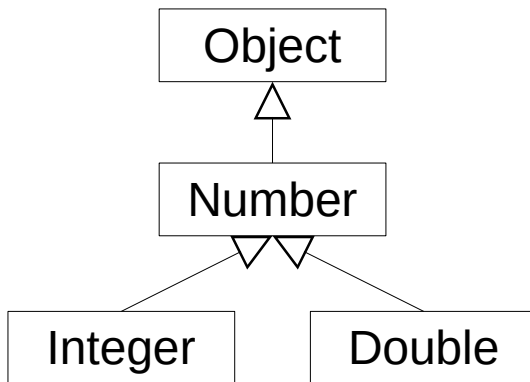
```
public class NumBox<T extends Number> {  
    private T obj  
  
    public int incAndGet() {  
        return obj.intValue() + 1;  
    }  
}
```





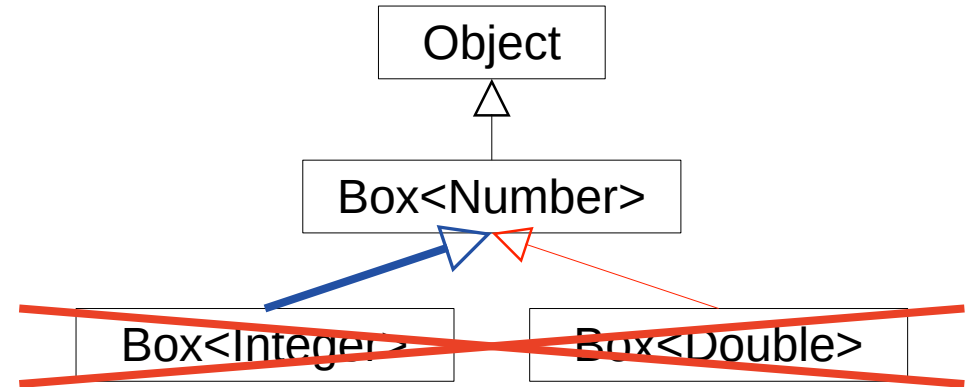
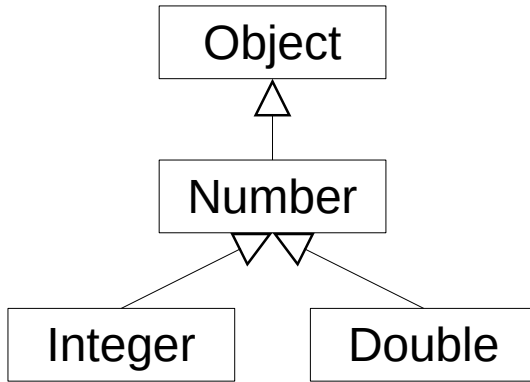
```
Number num;
Integer i = 511;
Double d = 3.1416;
num = i;
num = d;
```

```
Box<Number> bnum;
Integer i = 511;
Double d = 3.1416;
bnum = new Box<>(i);
bnum = new Box<>(d);
```

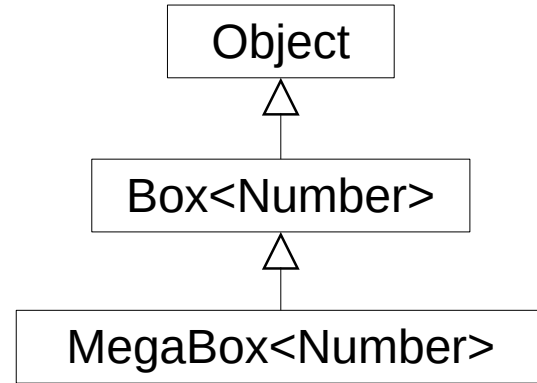
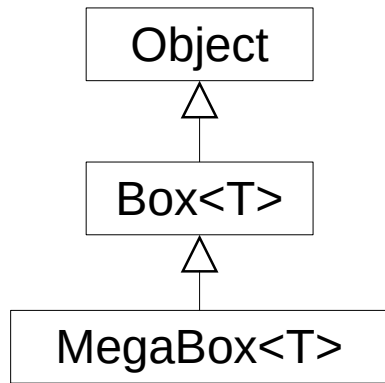


```
Box<Integer> bint;  
Integer i = 511;  
Double d = 3.1416;  
bint = new Box<>(i);  
bint = new Box<>(d); // cannot infer type arguments for Box<>  
bint = new Box<Double>(d); // Box<Double> cannot be converted to Box<Integer>  
bint = new Box<Integer>(d); // Double cannot be converted to Integer
```





```
Box<Integer> bint;
Integer i = 511;
Double d = 3.1416;
Box<Number> bnum = (Box<Number>) bint;
// Box<Integer> cannot be converted to Box<Number>
bnum = new Box<>(i);
bnum = new Box<>(d);
```



```
Box<Number> bnum;  
MegaBox<Number> mega; // MegaBox extends Box
```

```
mega = new MegaBox<>(3, 1415926535897932384626433832795);  
bnum = mega;
```

Обобщенный класс - параметр метода

```
public class NBox<N extends Number> extends Box<N> {  
  
    public NBox(N o) { super(o); }  
  
    public double avg(NBox<N> box) {  
        double v = box.get().doubleValue();  
        return (this.get().doubleValue() + v) / 2.0;  
    }  
}
```

```
NBox<Double> box1 = new NBox<>(1.0);  
NBox<Double> box2 = new NBox<>(5.0);  
System.out.println(box1.avg(box2));
```

Обобщенный класс - параметр метода

```
public class NBox<N extends Number> extends Box<N> {  
  
    public NBox(N o) { super(o); }  
  
    public double avg(NBox<N> box) {  
        double v = box.get().doubleValue();  
        return (this.get().doubleValue() + v) / 2.0;  
    }  
}
```

```
NBox<Double> box1 = new NBox<>(1.0);  
NBox<Integer> box2 = new NBox<>(5);  
System.out.println(box1.avg(box2));
```

error: incompatible types: NBox<Integer>
cannot be converted to NBox<Double>

Обобщенный класс - параметр метода

```
public class NBox<N extends Number> extends Box<N> {  
  
    public NBox(N o) { super(o); }  
  
    public double avg(NBox<? extends Number> box) {  
        double v = box.get().doubleValue();  
        return (this.get().doubleValue() + v) / 2.0;  
    }  
}
```

```
NBox<Double> box1 = new NBox<>(1.0);  
NBox<Integer> box2 = new NBox<>(5);  
System.out.println(box1.avg(box2));
```


- `Box<T>`
- `Box<?>`
- `Box<? extends Number>`
- `Box<Number>`
- `Box<Object>`
- `Box`

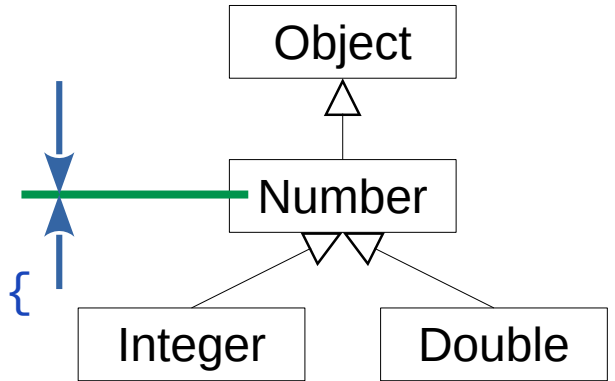
- `class List<T>`
 - ❖ `public double sum(List<? extends Number> list)`
 - ❖ `public void print(List<?> list)`
 - ❖ `public <T> void fill(List<? super T> list, T value)`
- `Class java.lang.Enum<E extends Enum<E>>`
 - ❖ `enum Color ~> class Color extends Enum<Color>`
 - ❖ Enum - это обобщенный класс, параметром которого является класс, который может наследоваться только от Enum с таким же параметром.

Ограничения снизу и сверху - методы

```
public class Box<T> {
    private T obj;

    public void copyTo(Box<? super T> to) {
        to.obj = obj;
    }
    public void copyFrom(Box<? extends T> from) {
        obj = from.obj;
    }
}

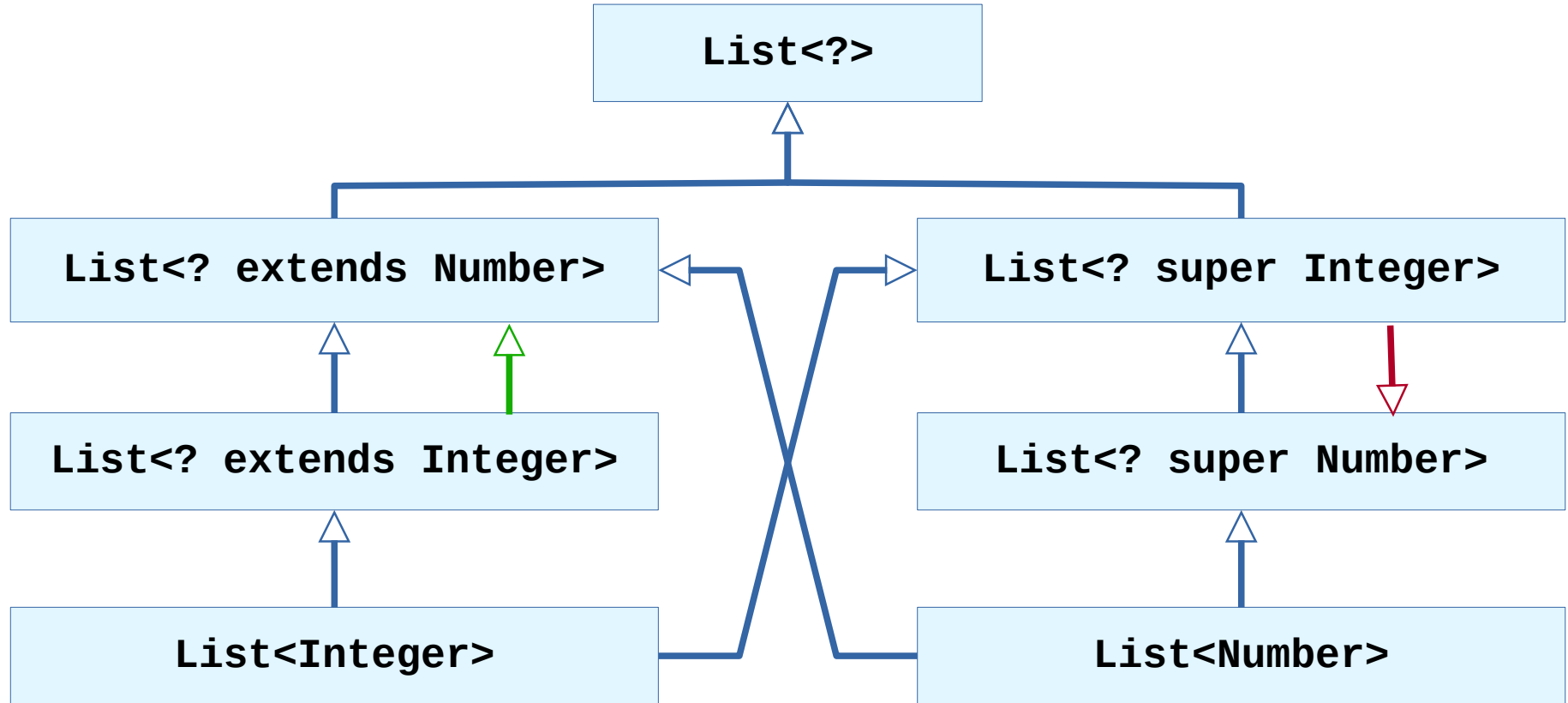
Box<Number> nbox = new Box<>();
Box<Integer> ibox = new Box<>(42);
ibox.copyTo(nbox);    // T = Integer, ? = Number
nbox.copyFrom(ibox); // T = Number, ? = Integer
```



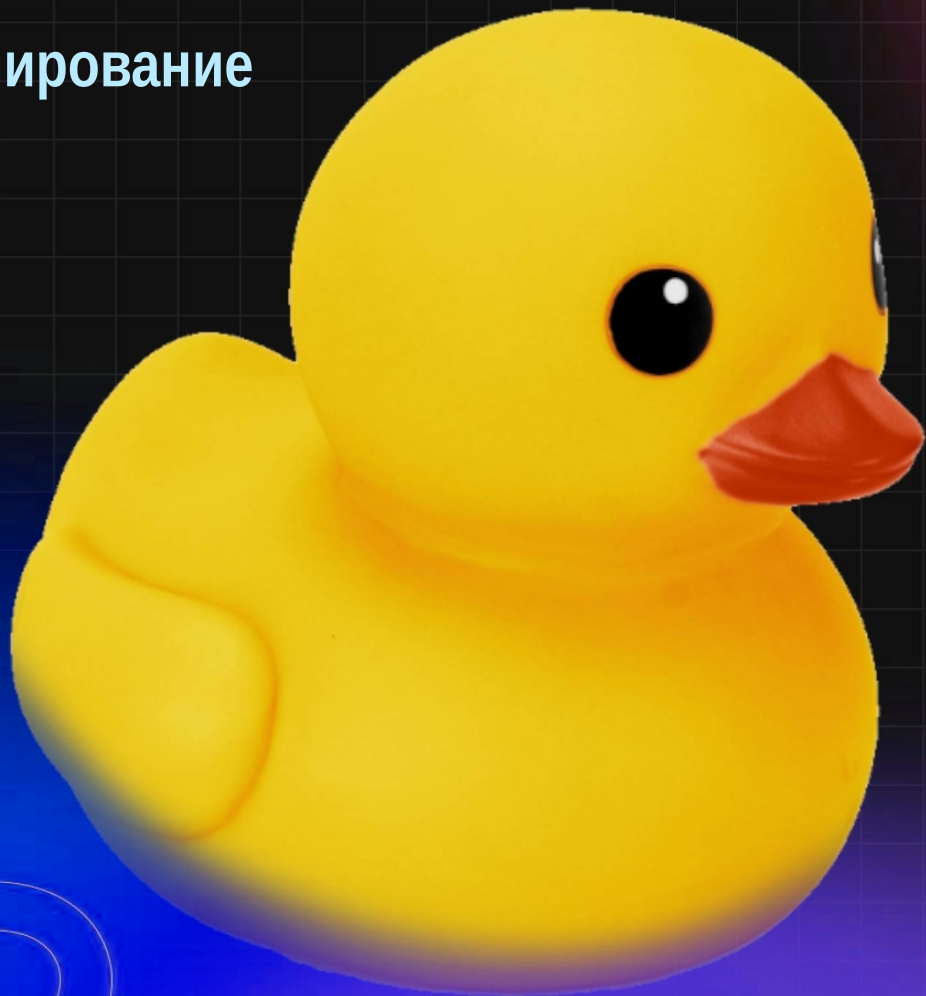
- Параметры методов
 - ❖ производители (producers)
 - ❖ потребители (consumers)
- Правило PECS - Producer Extends, Consumer Super
- Производители — ограничение сверху `<? extends X>`
- `<? extends Object> == <?>`
- Потребители — ограничение снизу `<? super X>`
- Параметр одновременно потребитель и производитель — подстановки не используются (тип задается явно)

- Наследование массивов - ковариантно
 - ❖ `Integer[]` потомок `Number[]`
- Наследование обобщений - инвариантно
 - ❖ `Box<Integer>` не потомок и не предок `Box<Number>`
- Ограничение сверху - ковариантно
 - ❖ `Box<? extends Number>`
- Ограничение снизу - контрвариантно
 - ❖ `Box<? super Number>`

Схема наследования обобщений



Программирование
2 семестр
2024



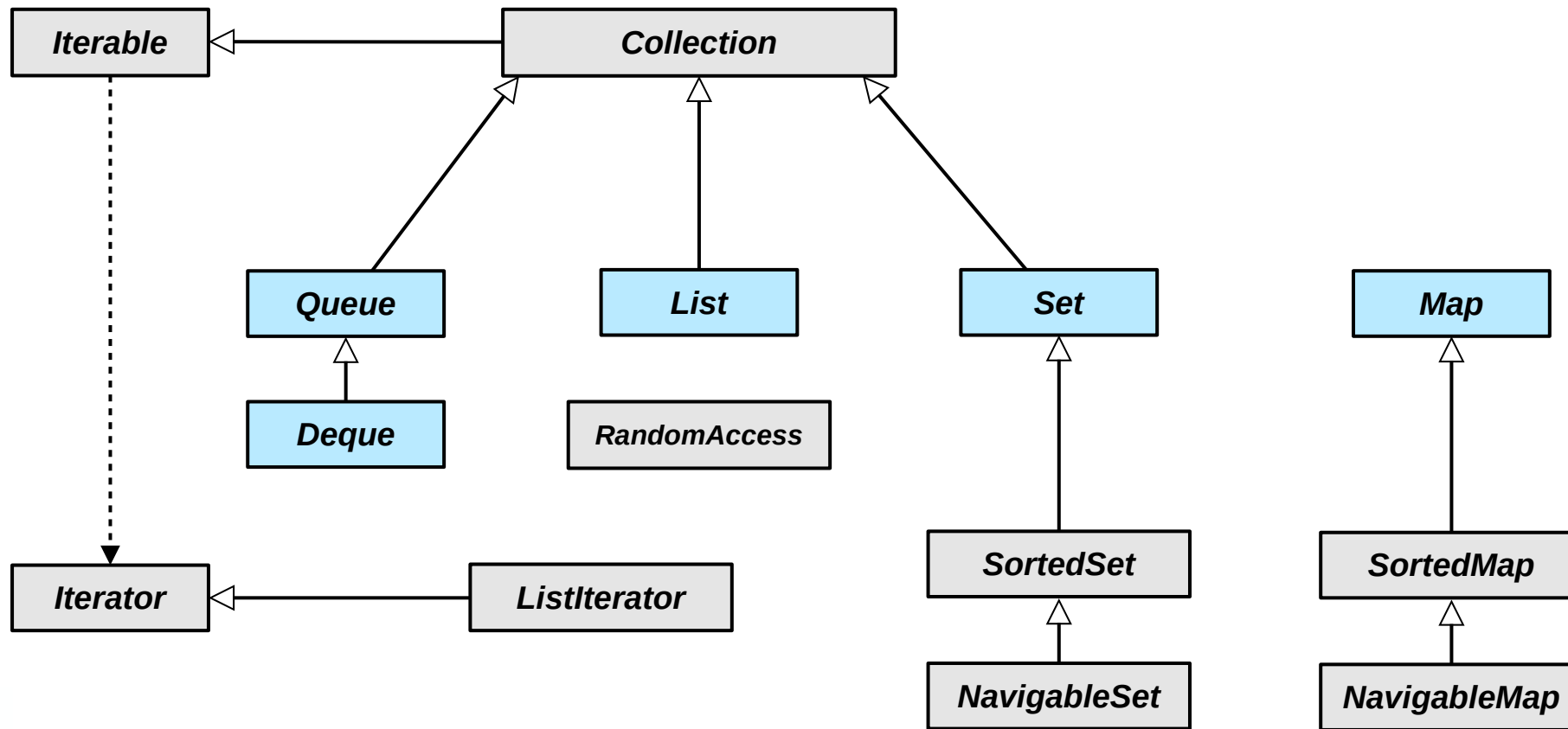
ІТМО

Коллекции

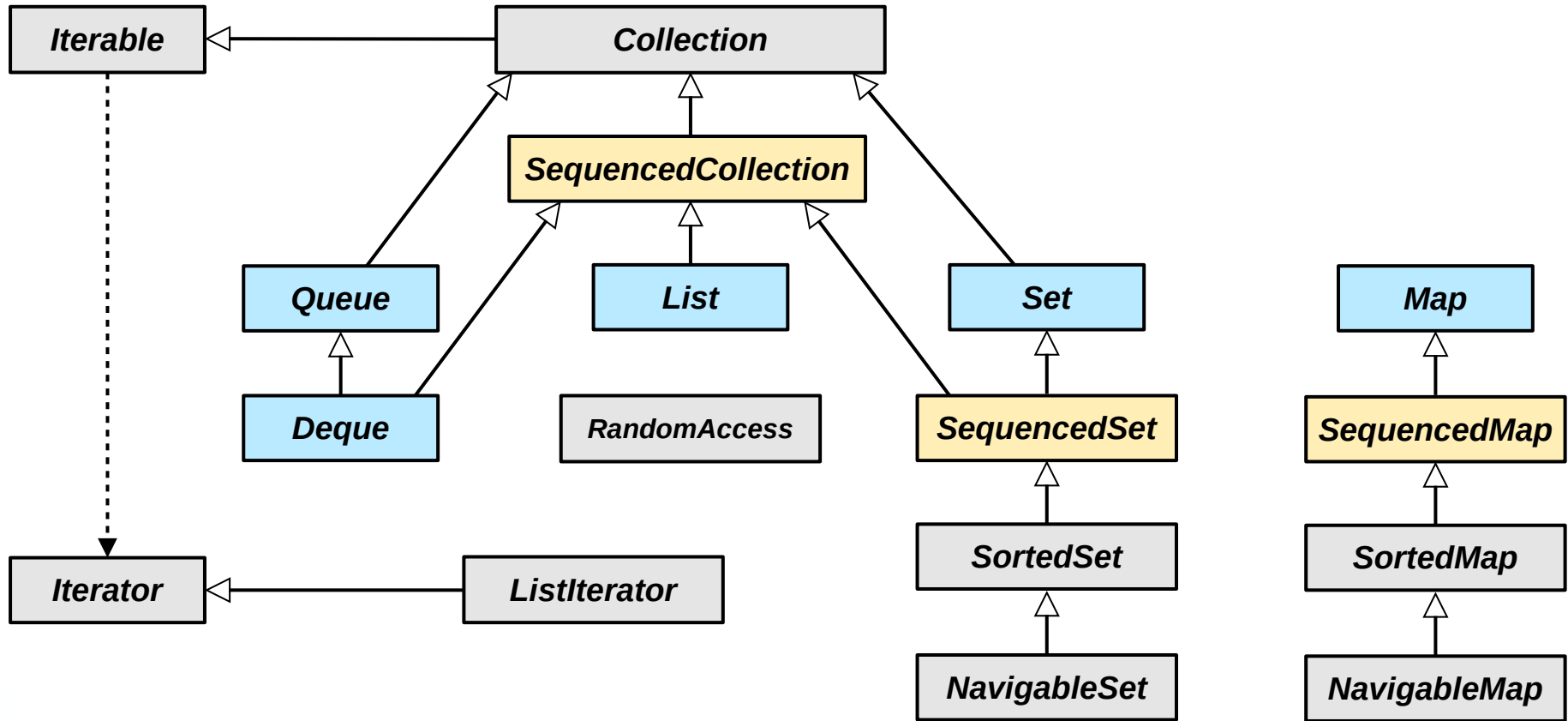
- Низкоуровневый тип
- Элементы индексированы (начиная с 0)
- Поддерживаются примитивные типы
- Размер нельзя изменить
- Быстрый доступ к любому элементу по индексу
 - ❖ $A_i = A_0 + i * size$
- Массив - Object (есть toString(), equals(), hashCode())

- Интерфейсы коллекций
- Абстрактные коллекции
- Реализации общего назначения
- Специальные реализации
- Потокбезопасные реализации
- Блокирующие очереди
- Вспомогательные классы

- Базовые интерфейсы определяют контракт
 - ❖ Что и как можно делать с элементами
 - ❖ Queue, Deque, List, Set, SortedSet, Map, SortedMap, ...
- Базовые реализации определяют характеристики
 - ❖ Скорость, занимаемая память, упорядоченность
 - ❖ Array, Linked List, Hash Table, Tree, Heap, ...
 - ❖ Big O-нотация



Основные интерфейсы (JDK 21)



- Collection - коллекция элементов
- SequencedCollection - упорядоченная коллекция
- Map - отображение, пары "ключ-значение"
- Set - множество уникальных элементов
 - ❖ SequencedMap / Set - упорядоченное отображение / множество
 - ❖ SortedMap / Set - отсортированное отображение / множество
 - ❖ NavigableMap / Set - "обходимое" отображение / множество
- List - индексированный список
- Queue - очередь
- Deque - двусторонняя очередь

- Queue - КЬЮ - очередь
- Deque - ДЕК - Double-Ended Queue - двусторонняя очередь
 - ❖ dequeue - декью - удалить из очереди
 - ❖ enqueue - енкью - поставить в очередь

- List : ArrayList vs LinkedList
- Set : HashSet vs LinkedHashSet vs TreeSet
- Map : HashMap vs LinkedHashMap vs TreeMap
- Queue : PriorityQueue vs LinkedList vs ArrayDeque

- Big-O нотация - асимптотическая оценка эффективности
 - ❖ $O(1)$
 - ❖ $O(\log n)$
 - ❖ $O(n)$
 - ❖ $O(n \log n)$
 - ❖ $O(n^2)$
 - ❖ $O(2^n)$

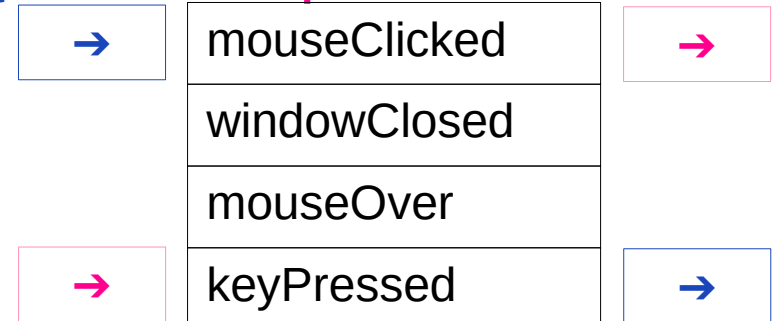
- List

0	яблоко
1	апельсин
2	банан
3	КИВИ

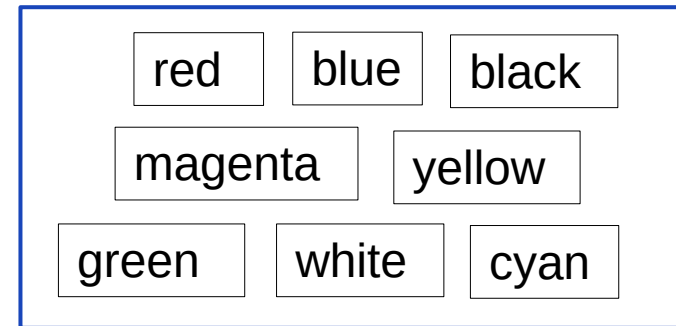
- Map

и	37153142
т	31620970
м	16203060
о	55414481

- Queue/Deque/Stack



- Set



- Итерируемый объект - **возможен обход** всех элементов
- Цикл `for (x : Iterable)`
- `Iterator<T> iterator()`

- Реализует обход элементов коллекции
- Методы:
 - ❖ boolean hasNext()
 - ❖ E next()
 - ❖ void remove()

- Пример

```
Iterator<Short> it = c.iterator();  
while (it.hasNext()) {  
    System.out.println(it.next());  
}
```

- `boolean add (E e)`
 - ❖ результат - **элемент есть** в коллекции
 - ❖ `true` - коллекция изменилась
- `boolean remove (Object o)`
 - ❖ результат - **элемента нет** в коллекции
 - ❖ `true` - коллекция изменилась
- `boolean contains (Object o)`
 - ❖ `true` - элемент есть, `false` - элемента нет

- `addAll(Collection c)`
- `removeAll(Collection c)`
- `retainAll(Collection c)`
- `containsAll(Collection c)`

- `clear()`
- `isEmpty()`
- `size()`

Интерфейс `SequencedCollection<E>`

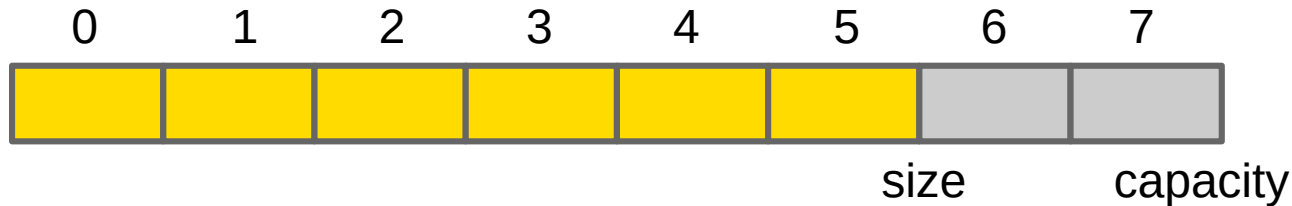
- `void addFirst(E e)`
- `void addLast(E e)`
- `E getFirst()`
- `E getLast()`
- `E removeFirst()`
- `E removeLast()`
- `SequencedCollection reversed()`

- Индекс от 0 до N
- E get(int index)
- E set(int index, E e)
- void add(int index, E e)
- E remove(int index)
 - ❖ List<Integer> list
 - ❖ list.remove(1) ???

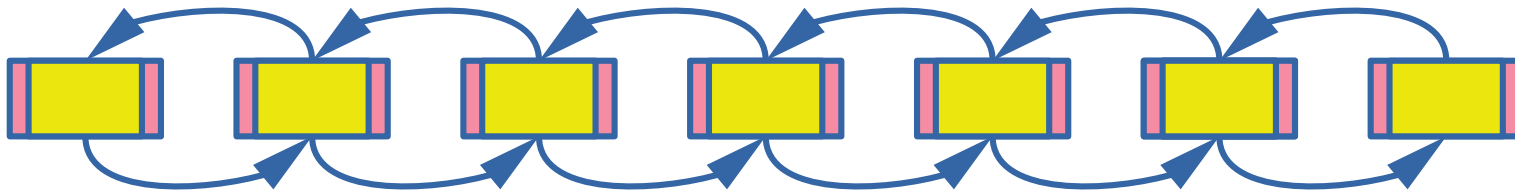
- Пример

```
List<String> list = new ArrayList<>();  
list.add("First element, index 0");  
list.add(list.get(0));  
var list2 = List<String>.of("X", "Y");  
list.add(list2);  
for (String s : list) {  
    System.out.println(s);  
}  
list.clear();
```

- Реализация списка (list) на основе массива (array)
- ArrayList extends AbstractList implements List, RandomAccess
- Быстрый произвольный доступ по индексу - $O(1)$



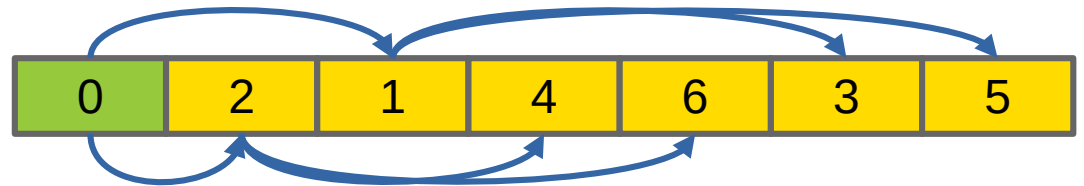
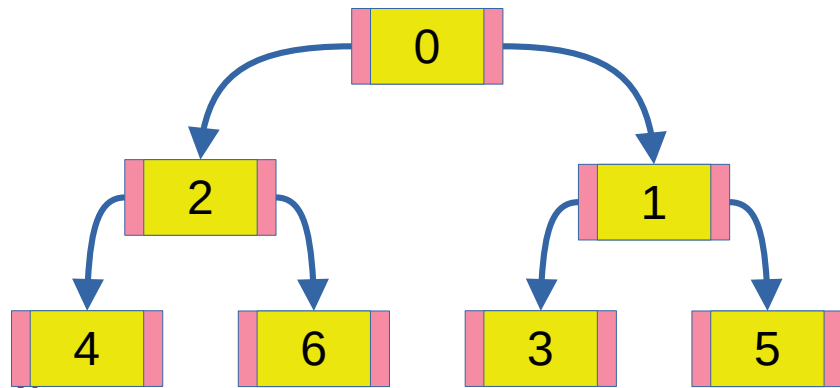
- Реализация на основе двусвязного списка
- LinkedList extends AbstractSequentialList implements List
- Последовательный доступ к элементам
- Быстрое добавление и удаление - $O(1)$



- Vector
 - ❖ потокобезопасный, но устаревший ArrayList
- Stack
 - ❖ устаревшая реализация на базе вектора
- CopyOnWriteArrayList
 - ❖ новая копия при каждом изменении

- Как правило - **FIFO (first in, first out)**
- Добавляем в конец, забираем из начала
- Методы, бросающие исключения
 - ❖ `boolean add()` , `E remove()`, `E element()`
- Методы, возвращающие особые значения (`null`)
 - ❖ `boolean offer()`, `E poll()`, `E peek()`

- Реализация на основе кучи (heap)
- Heap - дерево, корень - наименьший элемент
- FIFO + очередь тех, кто без очереди
- Нужна реализация сравнения элементов



- Двусторонняя очередь
- Может быть стеком
- Методы
 - ❖ Исключения
 - addFirst, removeFirst, getFirst
 - addLast, removeLast, getLast
 - ❖ null
 - offerFirst, pollFirst, peekFirst
 - offerLast, pollLast, peekLast

- Пример

```
Queue<String> queue = new ArrayDeque<>();  
queue.add("First element, index 0");  
queue.add("One more");  
var list = List<String>.of("X", "Y");  
queue.add(list);  
while (!queue.empty()) {  
    System.out.println(queue.remove());  
}
```

• Queue

❖ add(e)

❖ offer(e)

❖ remove()

❖ poll()

❖ element()

❖ peek()

• Deque

❖ addFirst(e)

❖ addLast(e)

❖ offerLast(e)

❖ removeFirst()

❖ pollFirst()

❖ getFirst()

❖ peekFirst()

• Stack

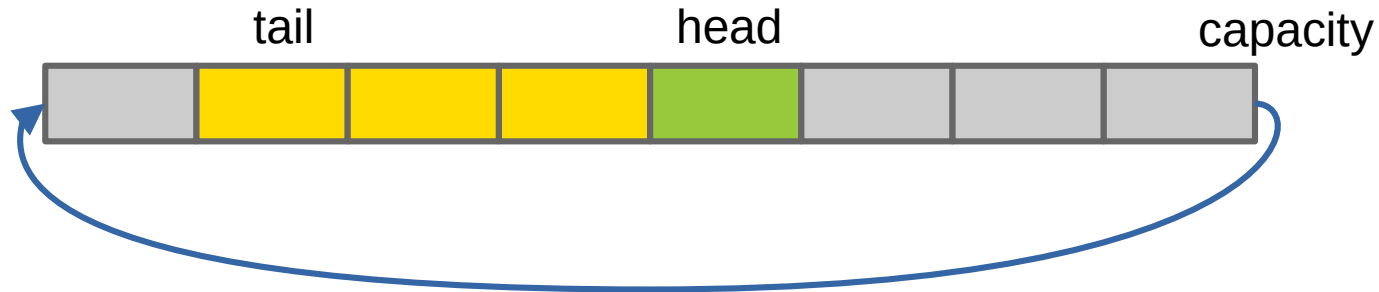
❖ push(e)

❖ pop()

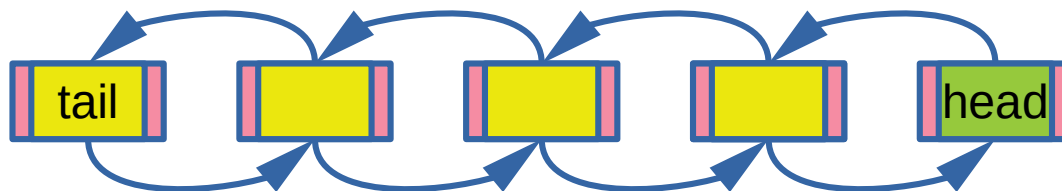
❖ peek()

Дек на основе массива - ArrayDeque

- Реализация на основе массива
- Быстрое добавление и удаление - $O(1)$



- LinkedList implements Queue, Deque, List
- Быстрые добавление, удаление - $O(1)$,



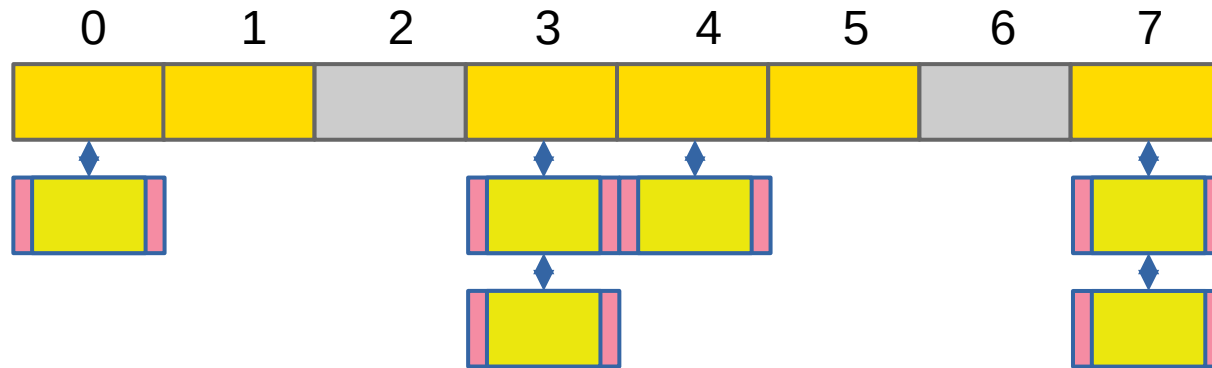
- Ассоциативный массив, отображение, словарь
- Два параметра типа
- Ключи уникальные, значения - любые
- Методы:
 - ❖ V put(K key, V value)
 - ❖ V get(Object K)
 - ❖ V remove(Object k)
- Представления:
 - ❖ keySet(), values(), entrySet()

- Пример

```
var map = new HashMap<String, Byte>();  
map.put("red", 100);  
map.put("green", 120);  
for (int i : map.values()) {  
    System.out.println(i);  
}
```


- Ассоциативный массив на основе хеш-таблицы
- $\text{bucket} = \text{key.hashCode()} \% N$
- Нужен метод `hashCode()` у элементов
- Обход элементов - в полном беспорядке

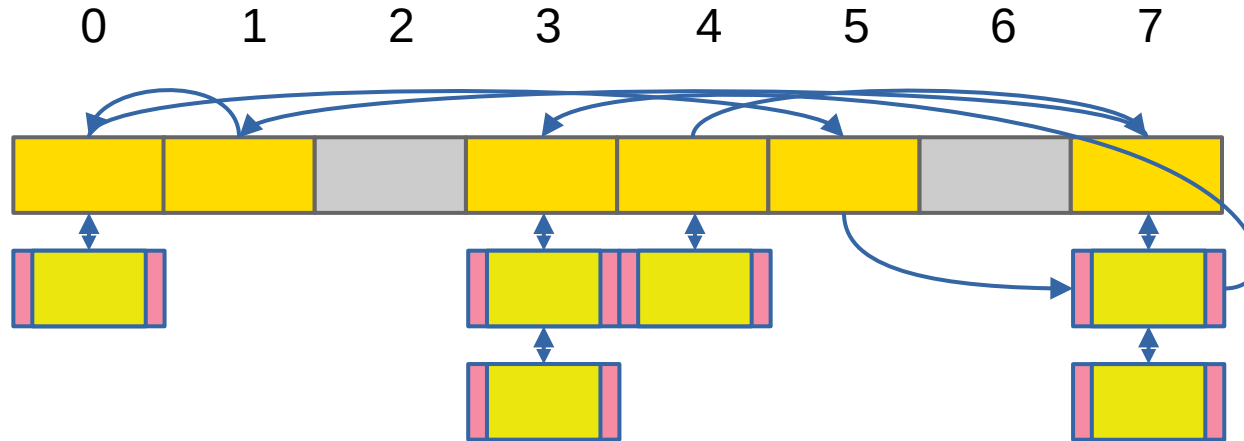
N buckets



SequencedMap + LinkedHashMap

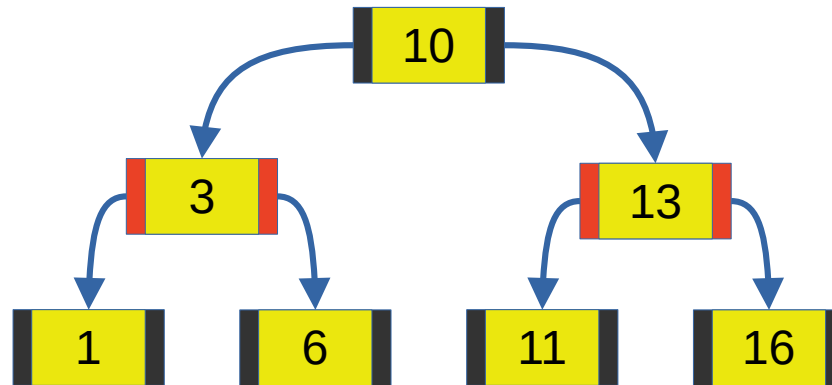
- Упорядоченность
- HashMap + связный список
- Получение элементов в порядке добавления

N buckets



- SortedMap<K, V> extends SequencedMap
 - ❖ K firstKey(), K lastKey()
 - ❖ SortedMap subMap(K from, K to)
 - ❖ SortedMap headMap(K k), SortedMap tailMap(K k)
- NavigableMap<K,V> extends SequencedMap, SortedMap
 - ❖ lowerKey, floorKey, ceilingKey, higherKey (<, <=, >= >)
 - ❖ lowerEntry, floorEntry, ceilingEntry, higherEntry
 - ❖ subMap, headMap, tailMap - (K key, boolean inclusive)

- Реализация на основе Red-Black Tree
- Самобалансирующееся бинарное дерево
- Элементы отсортированы
- Нужна реализация сортировки



- Hashtable
 - ❖ синхронизированная реализация
- WeakHashMap
 - ❖ ключи - слабые ссылки
- EnumMap
 - ❖ ключи - enum, реализация на основе массива
- IdentityHashMap
 - ❖ сравнение ключей по `==`, а не по `equals()`

- Методы как у Collection
- Массовые операции - объединение, пересечение, разность множеств
- Реализации - как у Map
 - ❖ HashSet implements Set,
 - ❖ LinkedHashSet extends HashSet implements SequencedSet,
 - ❖ TreeSet implements NavigableSet (SequencedSet, SortedSet)
 - ❖ Set реализован на основе Map

- CopyOnWriteArraySet
 - ❖ новая копия при изменении
- EnumSet
 - ❖ реализация на основе битовой карты

- List / Set

- ❖ .of()
- ❖ .of(E e1)
- ❖ .of(E e1, E e2)
- ❖ ... (до 10 элементов)
- ❖ .of(E... elements)
- ❖ .copyOf(Collection c)

- Map

- ❖ .of()
- ❖ .of(K k1, V v1)
- ❖ .of(K k1, V v1, K k2, V v2)
- ❖ ... (до 10 элементов)
- ❖ .ofEntries(Map.Entry... entries)
 - Map.Entry entry(K k, V v)
- ❖ .copyOf(Map m)

- коллекции-обертки (представления):
 - ❖ synchronized
 - ❖ unmodifiable
 - ❖ checked
- алгоритмы для List
 - ❖ sort(), shuffle(), reverse(), fill(), swap(), binarySearch()
- фабрики коллекций
 - ❖ emptySet, emptyList, emptyMap,
 - ❖ singleton, singletonList, singletonMap
- и еще много полезного (Queue asLifoQueue(Deque))

- Методы для работы с массивами (обычными)
 - ❖ сортировка, поиск, копирование, заполнение
 - ❖ и еще много полезного
 - ❖ `public static <T> List<T> asList(T... a)`
 - возвращает список на основе массива `a`
 - элементы менять можно
 - размер менять нельзя