УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Введение

ITMO more than a UNIVERSITY

Темы 2 семестра

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

Введение

- ☑ допуск на экзамен
  - 4 лабораторных * 9-13 (+2 ЛК) баллов = 36-60 баллов
  - 1 рубежка = 12-20 баллов
- ☑ 1 экзамен = 12-20 баллов
  - автомат - 70+ баллов перед экзаменом
- ☑ ИТОГО: **60-100** (+3 ЛК) баллов

Материалы

- ☑ 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/

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Обобщения

ITMO more than a UNIVERSITY

Система типов ЯП

- ☑ 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 и динамическая типизация

- ☑ 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 и слабая типизация

- ☑ Сильная
  - минимум неявного приведения
  - запрет операций над несовместимыми типами
- ☑ Слабая
  - неявное приведение
  - операции с разными типами
  - JavaScript

Примитивные и ссылочные типы

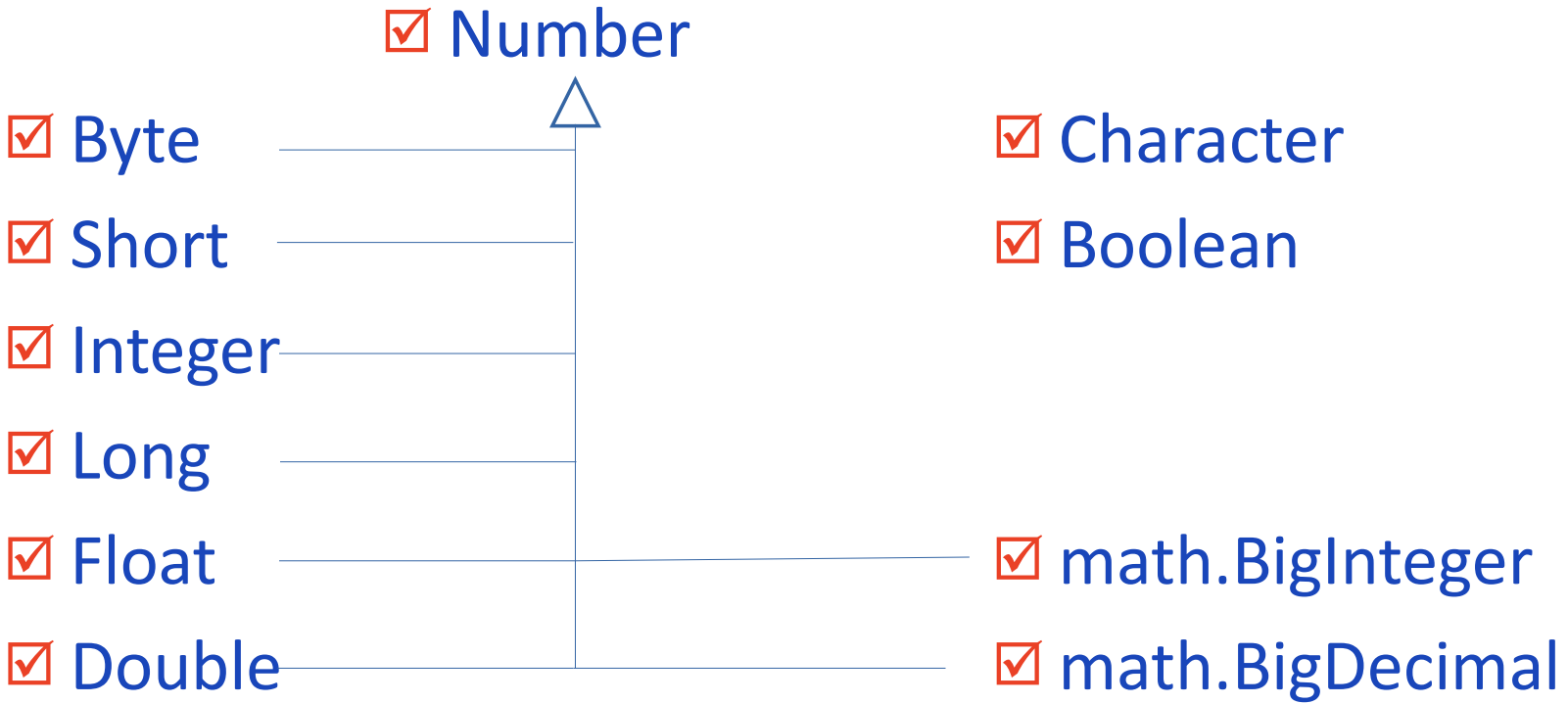
☑ Примитивные

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

☑ Ссылочные

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

Классы-оболочки (Wrappers)



Integer и int


```
~~Integer a = new Integer(10);~~
Integer b = Integer.valueOf(9999);
int c = b.intValue();

Integer x = 9111; // автоупаковка
int y = x;        // автораспаковка

x = x + 888;      // x = Integer.valueOf(x.intValue() + 888);

b == x           // false
b.equals(x)     // true
x = 100;
b.compareTo(x)    // что-то положительное (interface Comparable)
b = 100;
b == x           // true
```

Character и char

- ☑ 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)

Обобщения <Generics>

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

Пример без обобщений - Box


```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

Тест 1

УНИВЕРСИТЕТ ИТМО

```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box("Hello!");

int len = box.get().length();
```

Тест 1 - get() возвращает Object


```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box("Hello!");

int len = box.get().length();
```

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

Тест 2

УНИВЕРСИТЕТ ИТМО

```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box("Hello!");

String o = box.get();
int len = o.length();
```

Тест 2 - несовместимые типы

УНИВЕРСИТЕТ ИТМО

```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box("Hello!");

String o = box.get();
int len = o.length();
```

```
error: incompatible types: Object
cannot be converted to String
                String o = box.get();
                                   ^
```

Тест 3


```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box("Hello!");;

String o = (String) box.get();
int len = o.length();
```

Тест 3 - успешная компиляция

УНИВЕРСИТЕТ ИТМО

```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box(256);;

String o = (String) box.get();
int len = o.length();

// компиляция без ошибок!
```

Тест 3 - исключение приведения типа

УНИВЕРСИТЕТ ИТМО

```
public class Box {

  private Object obj;

  public Box(Object o) {
    obj = o;
  }

  public Object get() {
    return obj;
  }
}
```

```
Box box = new Box(256);

String o = (String) box.get();
int len = o.length();
```

```
Exception in thread "main"
java.lang.ClassCastException:
class java.lang.Integer cannot be cast
to class java.lang.String
```

Пример с обобщениями

УНИВЕРСИТЕТ ИТМО

```
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;
  }
}
```

Тест 4 - успешная компиляция

УНИВЕРСИТЕТ ИТМО

```
public class Box<T> {

  private T obj;

  public Box(T o) {
    obj = o;
  }

  public T get() {
    return obj;
  }
}
```

```
Box<String> box =
            new Box<>("Hello");

String o = (String) box.get();
int len = o.length();
```

Обобщенный тип (generic)

Параметризованный тип (parametrized)

Тест 5


```
public class Box<T> {

  private T obj;

  public Box(T o) {
    obj = o;
  }

  public T get() {
    return obj;
  }
}
```

```
Box<String> box =
                new Box<>(256);

String o = box.get();
int len = o.length();
```

Тест 5 - несовместимые типы

УНИВЕРСИТЕТ ИТМО

```
public class Box<T> {

  private T obj;

  public Box(T o) {
    obj = o;
  }

  public T get() {
    return obj;
  }
}
```

```
Box<String> box =
                new Box<>(256);

String o = box.get();
int len = o.length();
```

```
error: incompatible types: Integer
cannot be converted to String
                  box.put(s);
                             ^
```

Рекомендуемые параметры типа

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

Реализации обобщений и шаблонов

- ☑ 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

Стирание типов

☑ Совместимость
со старыми версиями

☑ Недостаточная
типобезопасность

- Контроль только
при компиляции
- При выполнении
возможны нарушения

☑ Нельзя делать:

- Обобщенный 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>)

Обобщенные методы

- ☑ Параметр типа перед типом возвращаемого значения

```
public static <T> Box<T> pack(T obj)
```

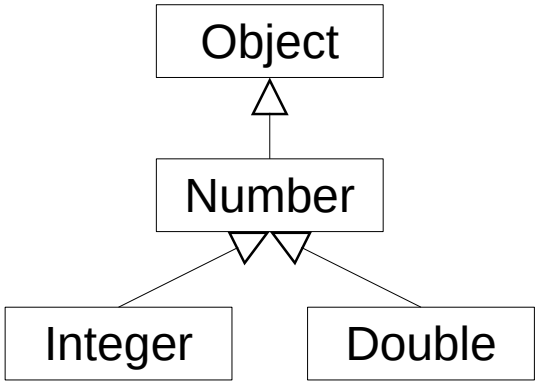
- ☑ При вызове метода нужен явный или неявный параметр

```
Box<Integer> b1 = Box.<Integer>pack(1);
Box<String> b2 = Box.pack("Hello");
```

Ограничение типа

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

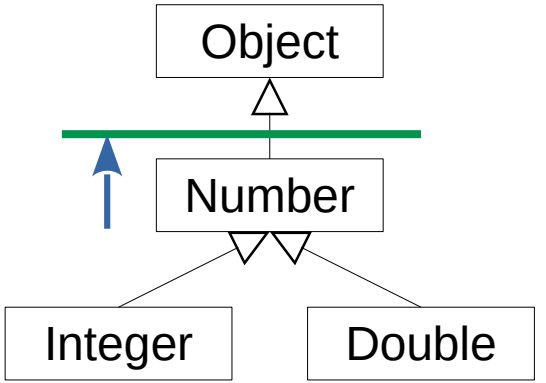
```
public class NumBox<T> {
  private Object obj

  public int incAndGet() {
    obj += 1;
    return obj;
  }
}
```



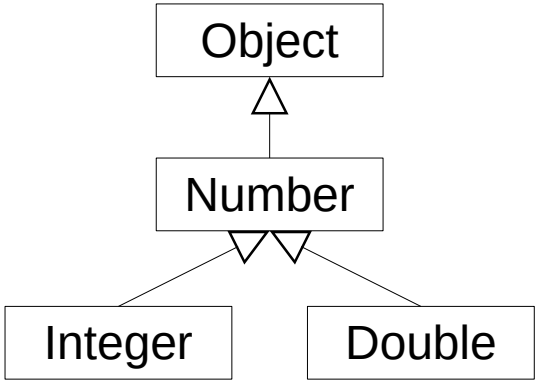
Ограничение сверху

- ☑ Инкремент только для чисел

```
public class NumBox<T extends Number> {
  private T obj

  public int incAndGet() {
    return obj.intValue() + 1;
  }
}
```

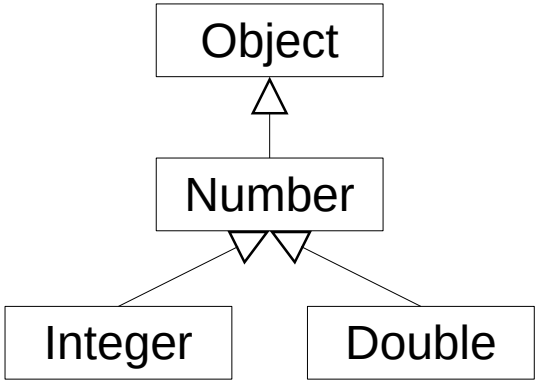


Обобщения и наследование



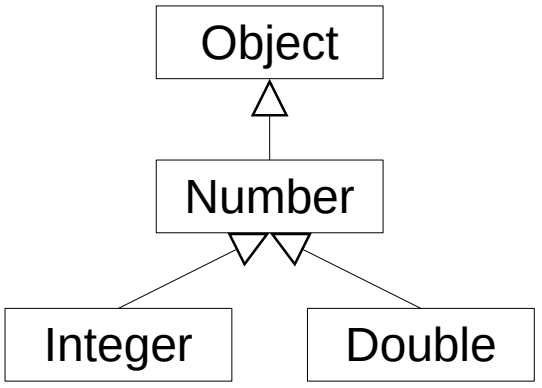

```
Number num;
Integer i = 511;
Double d = 3.1416;
num = i;
num = d;
```

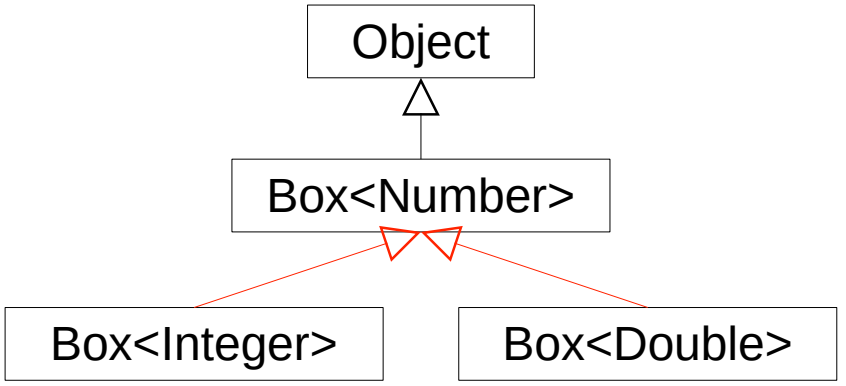
Наследование параметра типа




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

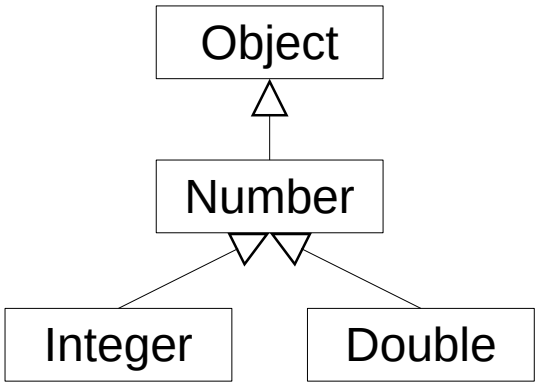
Наследование обобщений

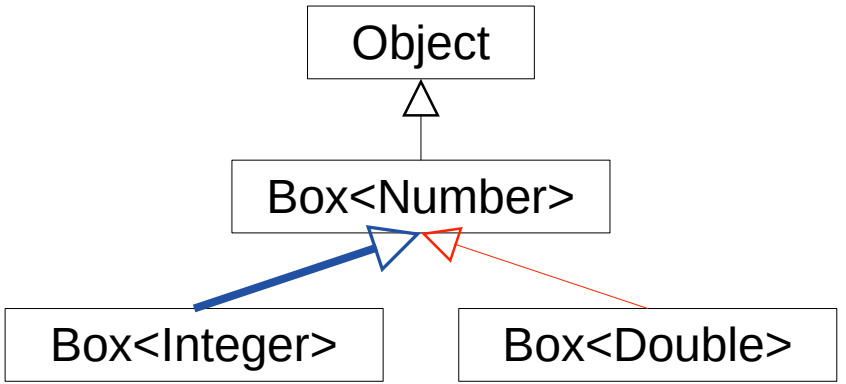





```
Box<Integer> bint;
Integer i = 511;
Double d = 3.1416;
bint = new Box<>(i);
bint = new Box<>(d);          // cannot infer type arguments for Box<>
bint = new Box<Double>(d); // Box<Double> cannot be converted to Box<Integer>
bint = new Box<Integer>(d);// Double cannot be converted to Integer
```

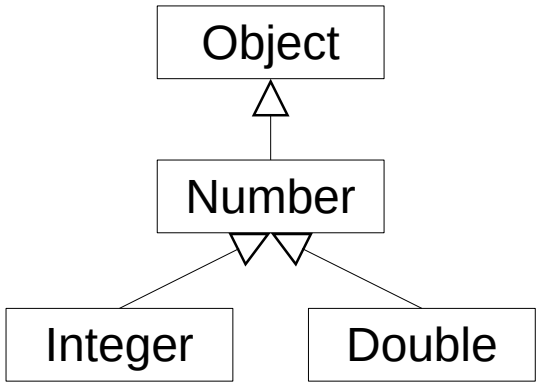
Тест 6

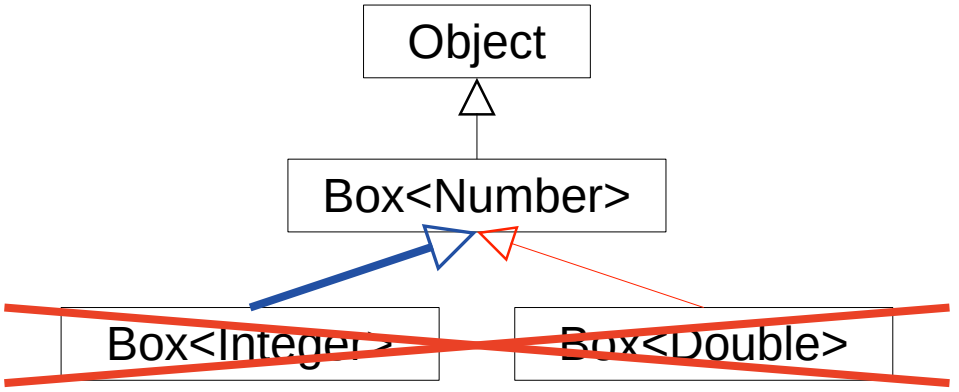





```
Box<Integer> bint;
Integer i = 511;
Double d = 3.1416;
Box<Number> bnum = (Box<Number>) bint;

bnum = new Box<>(i);
bnum = new Box<>(d);
```

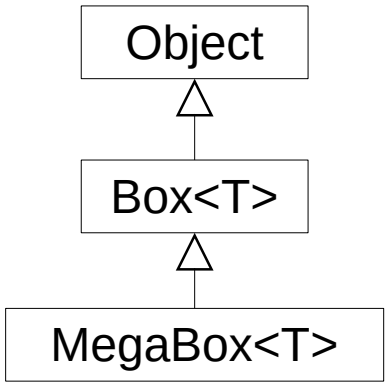
Тест 6 - несовместимые типы

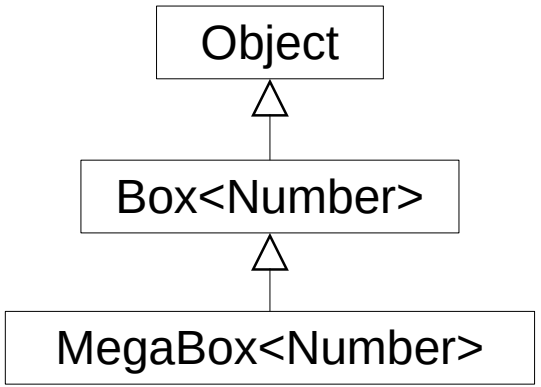





```
Box<Integer> bint;
Integer i = 511;
Double d = 3.1416;
Box<Number> bnum = (Box<Number>) bint;
                   //    Box<Integer> cannot be converted to Box<Number>
bnum = new Box<>(i);
bnum = new Box<>(d);
```

Обобщенные наследники






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

mega = new MegaBox<>(3,14159265358979323846264338327795);
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 (obj.doubleValue() + v) / 2.0;
  }
}

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

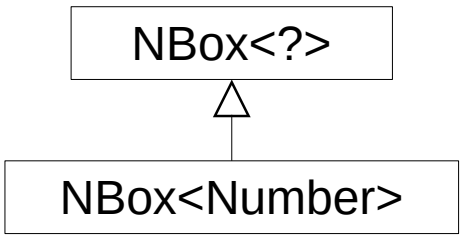
Обобщенный класс - параметр метода

УНИВЕРСИТЕТ ИТМО

```
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 (obj.doubleValue() + v) / 2.0;
  }
}

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

Подстановки (wildcards)


```
public class NBox<N extends Number> extends Box<N> {

  public double avg(NBox<? extends Number> box) {
    double v = box.get().doubleValue();
    return (obj.doubleValue() + v) / 2.0;
  }
}

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

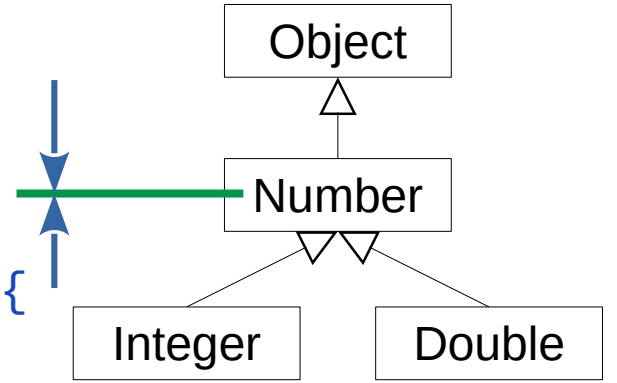


Виды подстановок

- ☑ 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)

Ограничения снизу и сверху


```
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
```



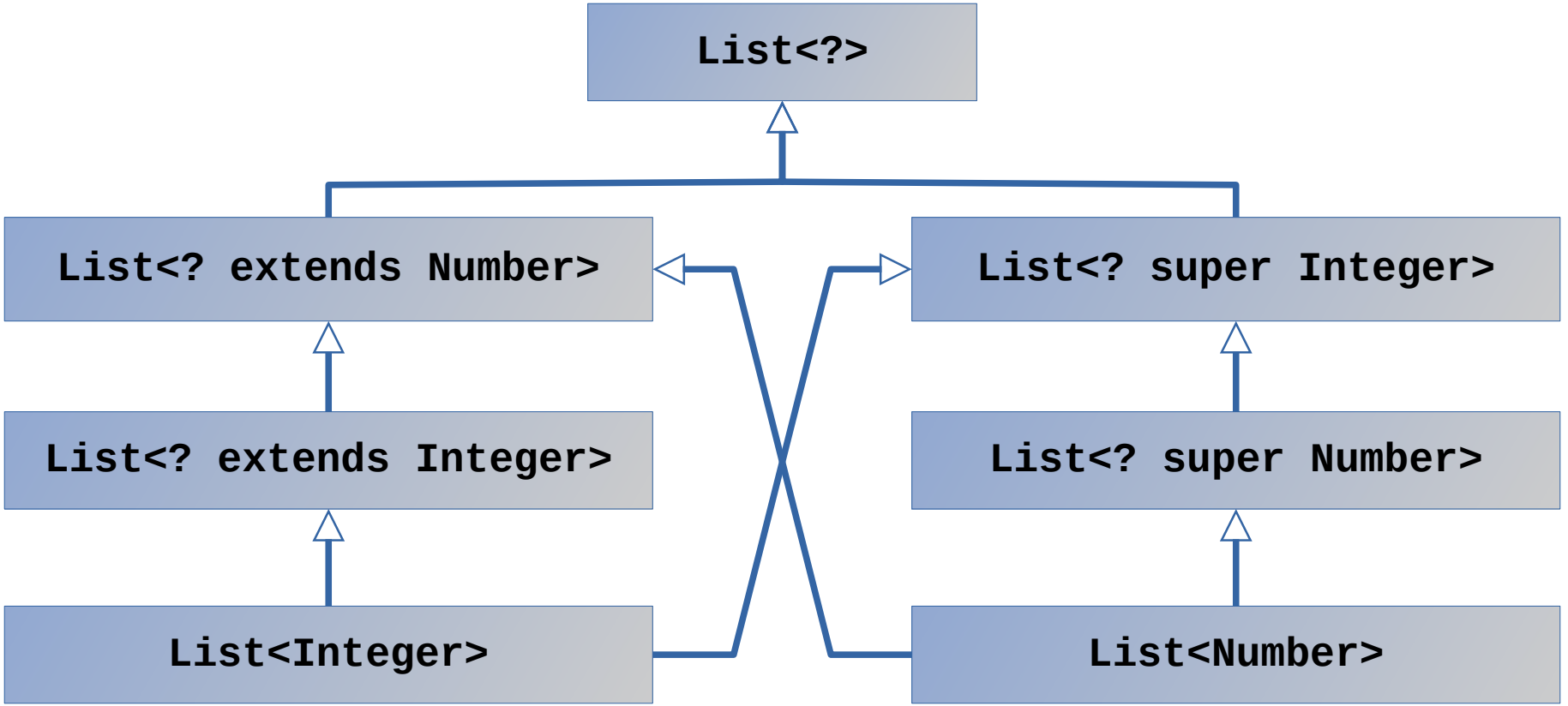
Правила ограничения

- ☑ Параметры методов - производители (producers) и потребители (consumers)
- ☑ Правило PECS - Producer Extends, Consumer Super
- ☑ Производители ограничиваются сверху <? extends X>
- ☑ Потребители ограничивается снизу <? super X>
- ☑ Производители, ограниченные классом Object - использовать неограниченную подстановку <?>
- ☑ Параметр одновременно потребитель и производитель — подстановки не используются (тип задается явно)

Обобщения и массивы

- ☑ 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



Java

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Коллекции

ITMOre than a UNIVERSITY

Collection Framework (java.util)

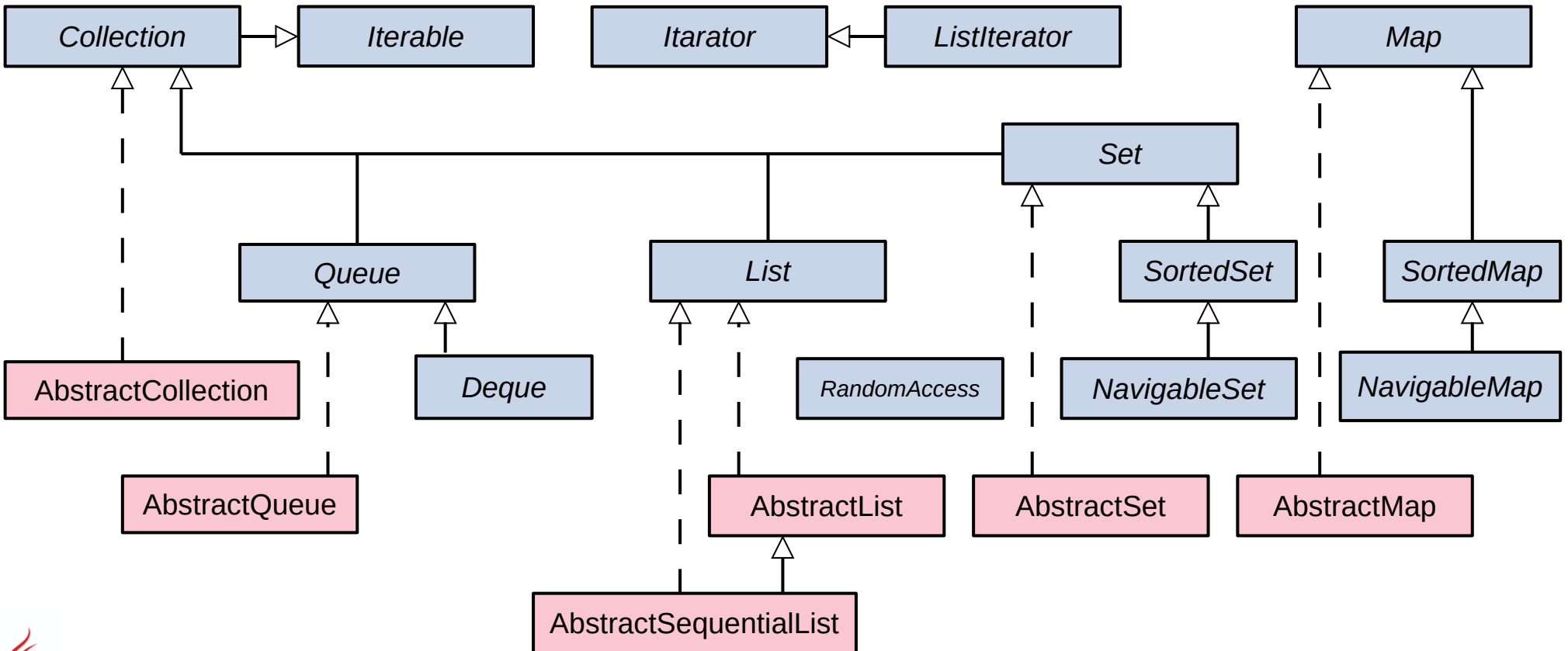
- ☑ Интерфейсы коллекций
- ☑ Абстрактные коллекции
- ☑ Реализации общего назначения
- ☑ Специальные реализации
- ☑ Потокобезопасные реализации
- ☑ Блокирующие очереди
- ☑ Вспомогательные классы

Принципы

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

Основные интерфейсы и абстракции

УНИВЕРСИТЕТ ИТМО



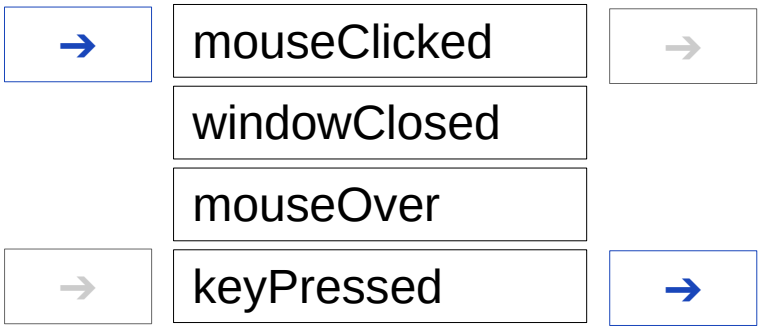
Интерфейсы - контракты

- ☑ Collection - коллекция одинаковых элементов
- ☑ Map - ассоциативный массив, пары "ключ-значение"
- ☑ SortedMap - сортировка по ключам
- ☑ List - последовательность индексированных элементов
- ☑ Set - множество уникальных элементов
- ☑ SortedSet - сортировка по значениям
- ☑ Queue - очередь, элементы входят и выходят
- ☑ Deque - двусторонняя очередь

List

0	яблоко
1	апельсин
2	банан
3	киви

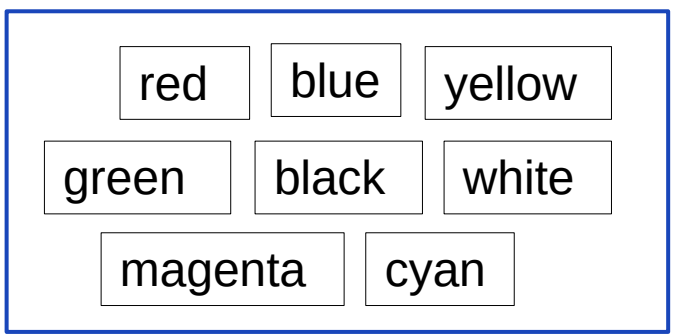
Queue/Deque/Stack



Map

и	37153142
т	31620970
м	16203060
о	55414481

Set



Интерфейс Iterable<T>

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

Интерфейс Iterator<E>

- ☑ Обход элементов коллекции
- ☑ Методы:
  - boolean hasNext()
  - E next()
  - void remove()

☑ Пример

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

Интерфейс Collection<E>

- ☑ 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

Массовые операции

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

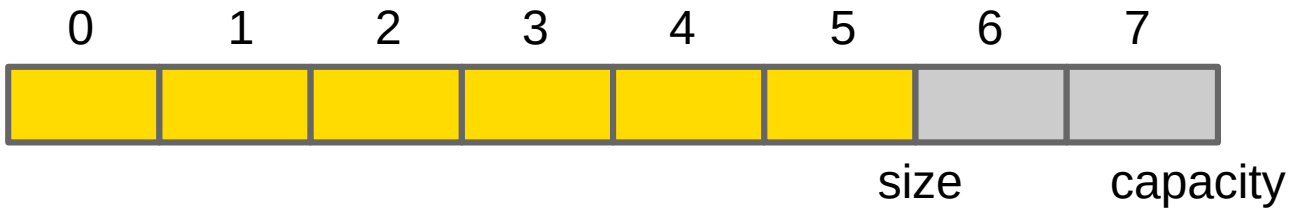
- ☑ clear()
- ☑ isEmpty()
- ☑ size()

Интерфейс List<E>

- ☑ 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) ???

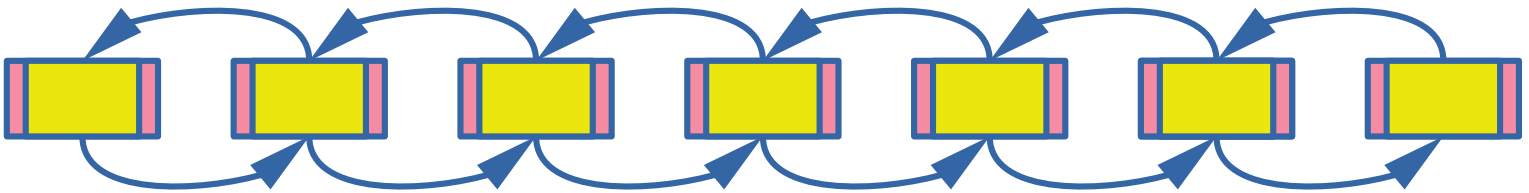
Динамический массив ArrayList

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



Двусвязный список LinkedList

- ☑ Реализация на основе двусвязного списка
- ☑ LinkedList extends AbstractSequentialList implements List
- ☑ Последовательный доступ к элементам
- ☑ Быстрое добавление и удаление



Еще списки

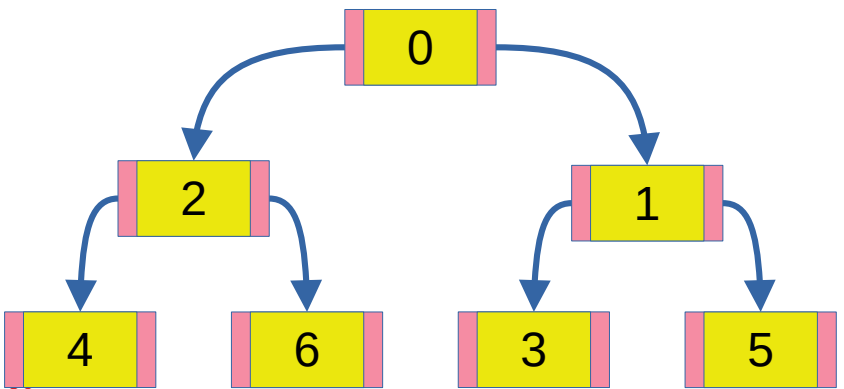
- ☑ Vector
  - потокобезопасный ArrayList
- ☑ Stack
  - устаревшая реализация на базе вектора
- ☑ CopyOnWriteArrayList
  - новая копия при каждом изменении

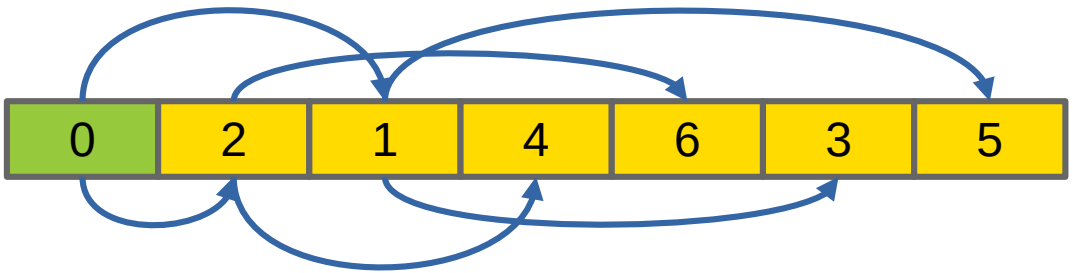
Интерфейс Queue<E>

- ☑ 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
  - boolean offer(), E poll(), E peek()

Приоритетная очередь PriorityQueue

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



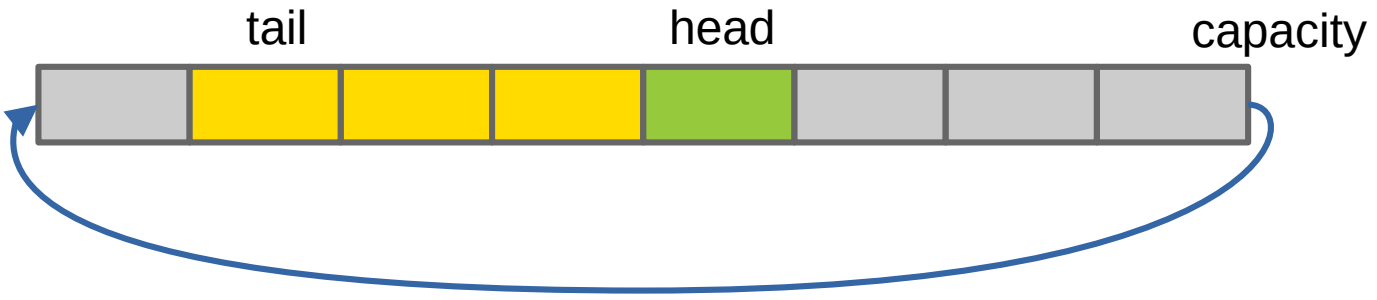


Интерфейс Deque<E>

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

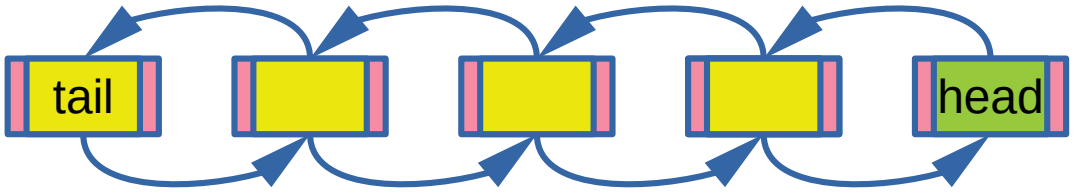
Дек на основе массива - ArrayDeque

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



И снова двусвязный список

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

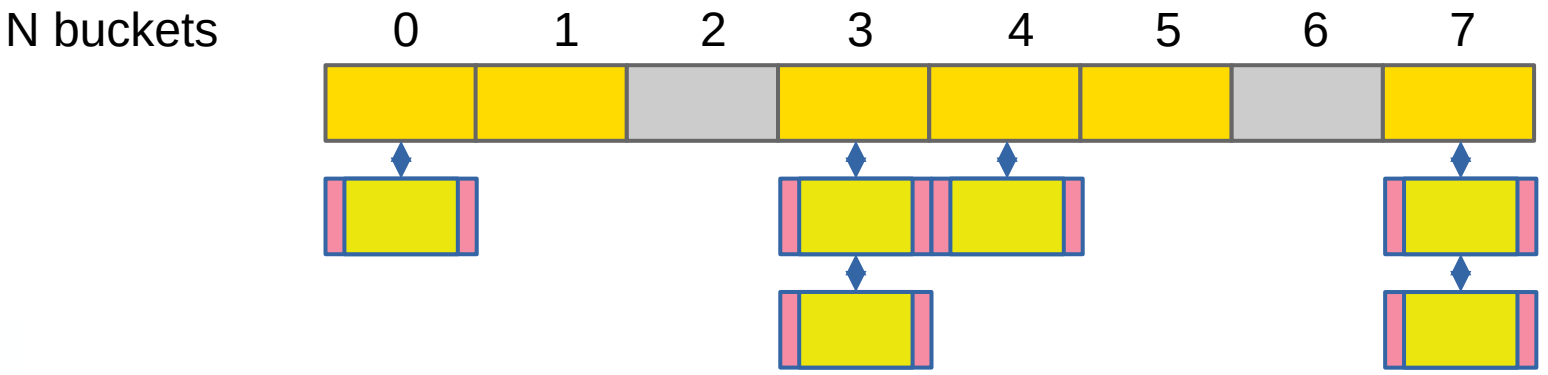


Интерфейс Map<K,V>

- ☑ 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()

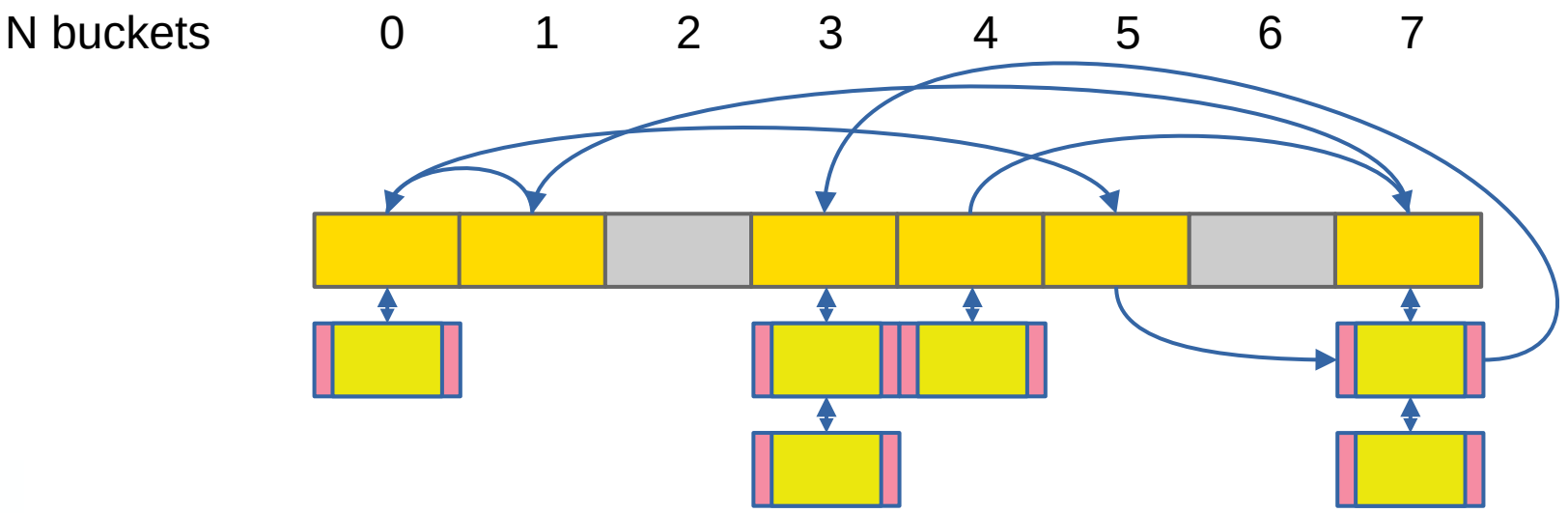
Хеш-таблица HashMap

- ☑ Ассоциативный массив на основе хеш-таблицы
- ☑ bucket = key.hashcode() % N
- ☑ Нужен метод hashCode() у элементов
- ☑ Обход элементов - в полном беспорядке



LinkedHashMap

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

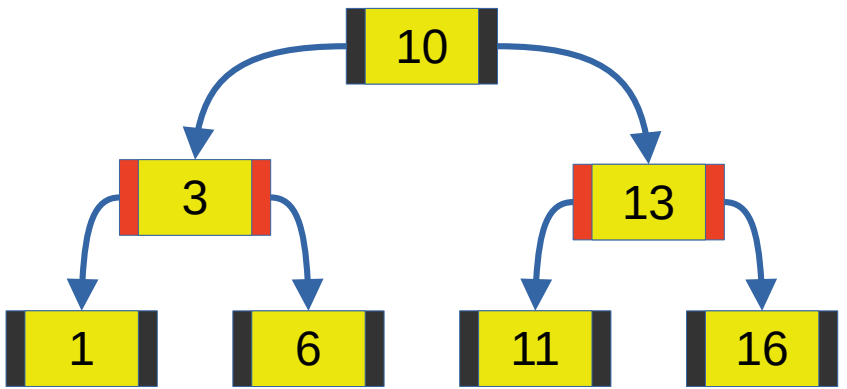


Интерфейсы SortedMap, NavigableMap

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

Дерево TreeMap

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



Еще словари

- ☑ Hashtable
  - синхронизированная реализация
- ☑ WeakHashMap
  - ключи - слабые ссылки
- ☑ EnumMap
  - ключи - enum, реализация на основе массива
- ☑ IdentityHashMap
  - ключи одинаковые только в случае идентичности

Интерфейс Set<E>

- ☑ Методы как у Collection
- ☑ Массовые операции - объединение, пересечение, разность множеств
- ☑ Реализации - как у Map
  - HashSet implements Set,
  - LinkedHashSet extends HashSet,
  - TreeSet implements SortedSet, NavigableSet
  - ключи являются значениями

Еще множества

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

Сравнение реализаций

- ☑ ArrayList vs LinkedList
- ☑ HashSet vs LinkedHashSet vs TreeSet
- ☑ HashMap vs LinkedHashMap vs TreeMap
- ☑ PriorityQueue vs LinkedList vs ArrayDeque

класс Collections

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

Класс Arrays

- ☑ Методы для работы с массивами (обычными)
  - сортировка, поиск, копирование, заполнение
  - и еще много полезного

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Сравнение элементов

ITMO more than a UNIVERSITY

Сравнение элементов

- ☑ Интерфейсы Comparable<T> и Comparator<T>
  - основной метод возвращает int (< 0, == 0, > 0)
- ☑ java.lang.Comparable<T> (x.compareTo(y))
  - естественный порядок сортировки
  - реализуется при создании класса
  - реализован в большинстве библиотечных классов
- ☑ java.util.Comparator<T> (c.compare(x, y))
  - любой необходимый порядок сортировки
  - объект создается по мере необходимости

Обобщенный алгоритм сортировки

1) Выбираем два элемента
2) Как-то сравниваем - Comparable или Comparator
3) При необходимости перемещаем элементы

☑ Готовые эффективные алгоритмы сортировки

☑ Все, что нужно - предоставить метод сравнения

Сортировка (естественная)

```
public class A {
  List<String> list;
  public void do() {
    Collections.sort(list);
    for (String s : list) { System.out.println(s); }
    }
}

class String implements Comparable
```

Сортировка (отдельный компаратор)

```
public class A {
  List<String> list;
  public void do() {
    Collections.sort(list, new C());
    for (String s : list) { System.out.println(s); }
    }
}
class C implements Comparator<String> {
  public int compare(String s1, String s2) {
    return s1.length() - s2.length();
  }
}
```

Сортировка (статический вложенный класс)

```
public class A {
  List<String> list;
  public void do() {
    Collections.sort(list, new C());
    for (String s : list) { System.out.println(s); }
  }
  static class C implements Comparator<String> {
    public int compare(String s1, String s2) {
      return s1.length() - s2.length();
    }
  }
}
```

Сортировка (внутренний класс)

```
public class A {
  List<String> list;
  public void do() {
    Collections.sort(list, new A().new C());
    for (String s : list) { System.out.println(s); }
  }
  class C implements Comparator<String> {
    public int compare(String s1, String s2) {
      return s1.length() - s2.length();
    }
  }
}
```

Сортировка (локальный класс)

```
public class A {
  List<String> list;
  public void do() {
    class C implements Comparator<String> {
      public int compare(String s1, String s2) {
        return s1.length() - s2.length();
      }
    }
    Collections.sort(list, new C());
    for (String s : list) { System.out.println(s); }
  }
}
```

Сортировка (анонимный класс)

```
public class A {
  List<String> list;
  public void do() {
    Collections.sort(list, new Comparator<String>() {
      public int compare(String s1, String s2) {
        return s1.length() - s2.length();
      }
    });
    for (String s : list) { System.out.println(s); }
  }
}
```

Сортировка (лямбда-выражение)

```
public class A {
  List<String> list;
  public void do() {
    Collections.sort(list,
      (s1, s2) -> s1.length() - s2.length());
    for (String s : list) { System.out.println(s); }
  }
}
```

УНИВЕРСИТЕТ ИТМО

Программирование. 2 семестр

Лабораторные

ITMO more than a UNIVERSITY

Лаба 5

- ☑ Внимательно прочитать задание и подумать
  - Посмотреть любой вариант лаб 6, 7 и 8
- ☑ Выделить классы (не забывать SOLID)
  - Подумать над общей структурой
- ☑ Нарисовать диаграмму
  - Объяснить уточке, как это должно работать
- ☑ Написать маленький работающий код
  - Дописывать до победы

Команды

- ☑ Command (getName(), getDescr(), execute())
  - HelpCommand
  - ShowCommand
  - AddCommand
- ☑ Map<String, Command>
  - map.put(command.getName(), command)

    ...
  - map.get(str[0]).execute();