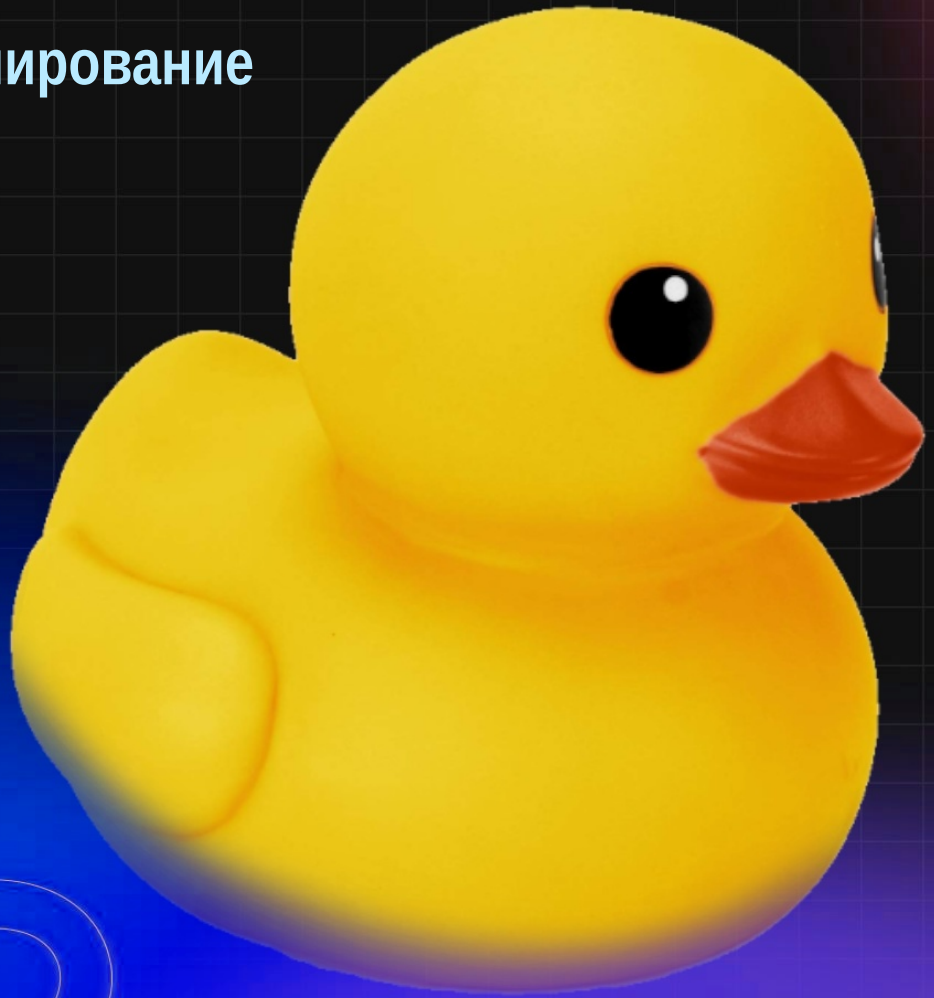
Программирование
2 семестр
2025

ИТМО



Введение

Гаврилов Антон Валерьевич

Темы 2 семестра

ИТМО

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

Введение

ИТМО

- Гугложурнал
  - лаба 5 (60%...100%)
  - лабы 6 и 7 (60%...100%)
  - рубеж (60%...100%)

    = допуск
  - лаба 8 (60%...100%)

    (условно на 4 или 5)
  - экзамен (60%...100%)
- БаРС
  - лаба 1 (12...15)
  - лаба 2 (24...30)
  - рубеж (12...20)

    автомат 3E/3D, если >= 60
  - + 5 / 10 к максимуму за Л1 / 2

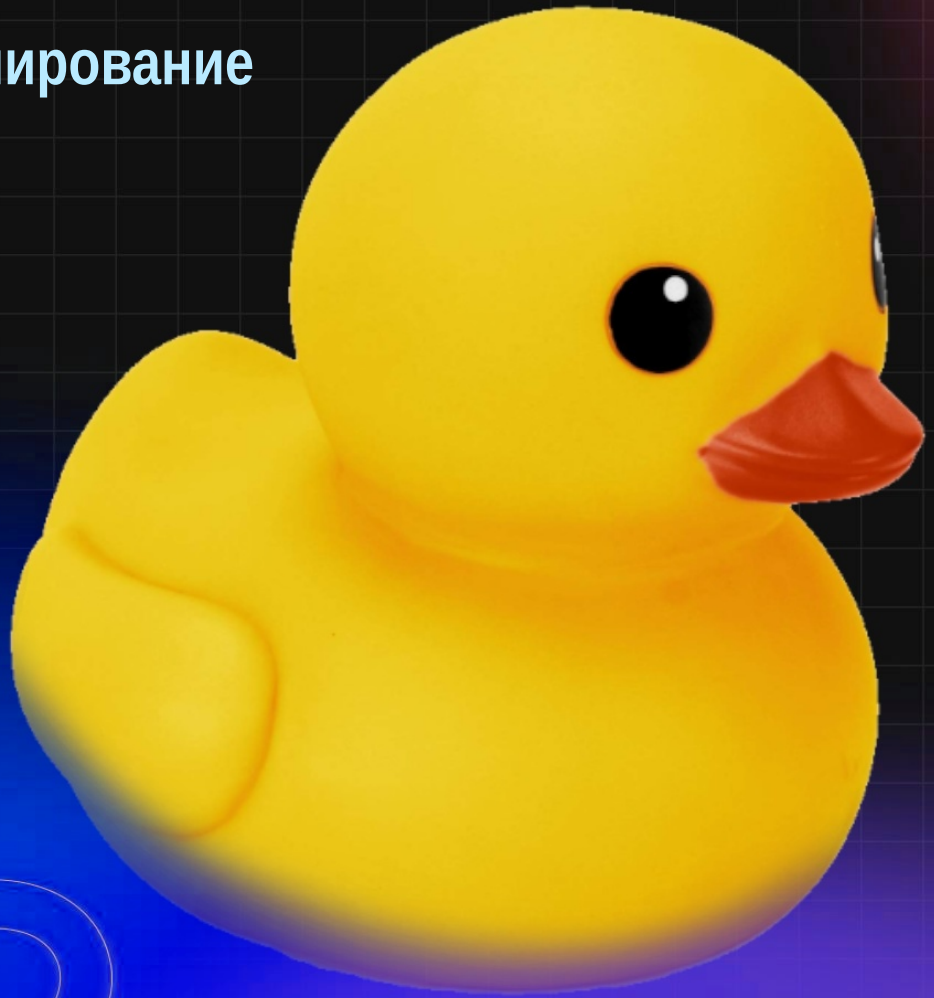
    автомат 5A, если > 70
  - экзамен (12...20) [+ 3 ЛК]

Материалы

- https://se.ifmo.ru
  - задания к лабораторным
  - видео и тексты лекций
  - методички
  - вопросы к экзамену

- 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
2025



Типы данных
Обертки
(Wrappers)

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

ИТМО

- 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

Java

Сильная и слабая типизация

ИТМО

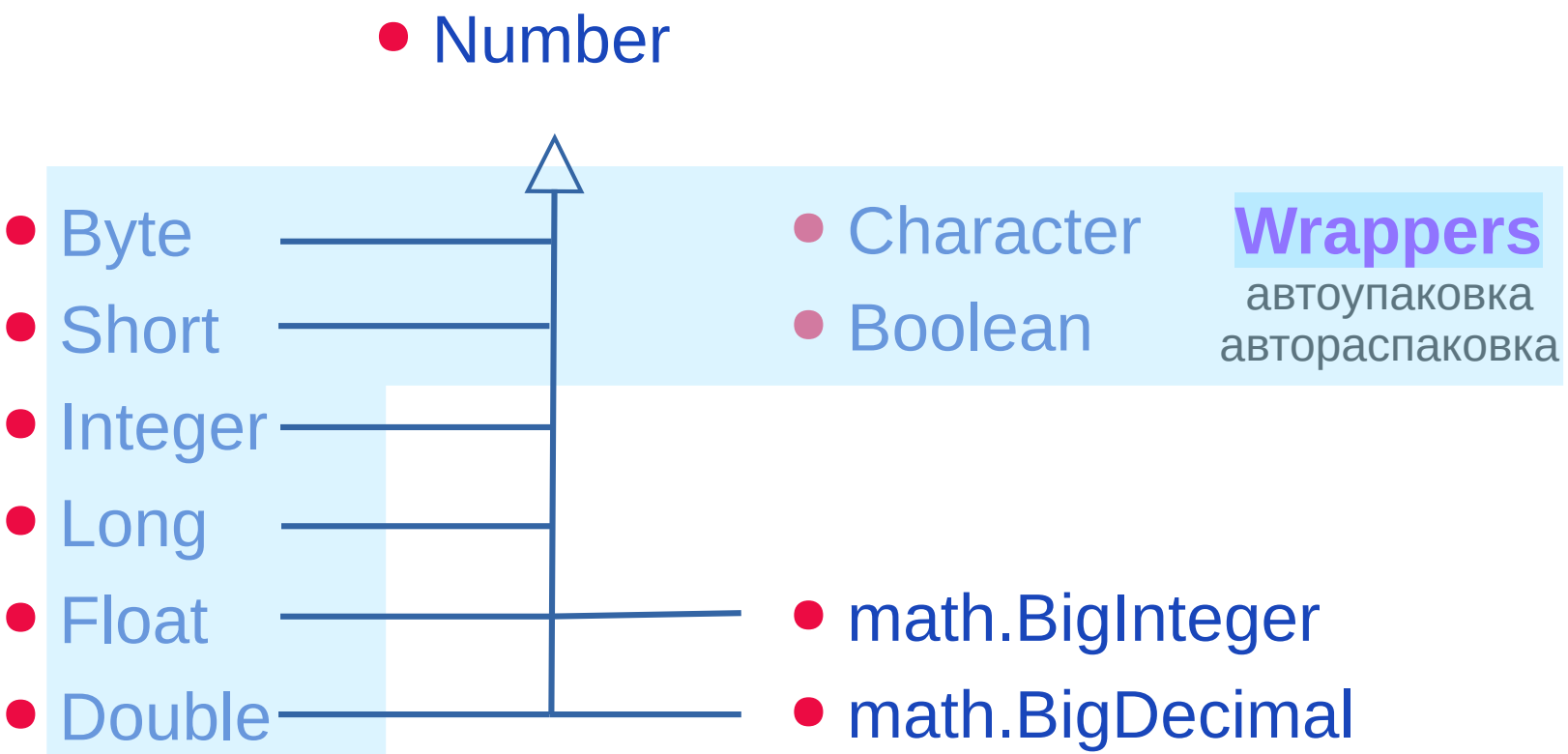
- Сильная
  - минимум неявного приведения
  - запрет операций над несовместимыми типами
- Слабая
  - неявное приведение
  - операции с разными типами
  - JavaScript
    - https://www.destroyallsoftware.com/talks/wat
      (C) Gary Bernhardt, 2012

Примитивные и ссылочные типы

ИТМО

- Примитивные
  - значение
  - не наследник Object
  - эффективные
  - операции
  - массивы
  - не обобщается
- Ссылочные
  - ссылка
  - наследник Object
  - больше и медленнее
  - методы
  - массивы и коллекции
  - обобщается

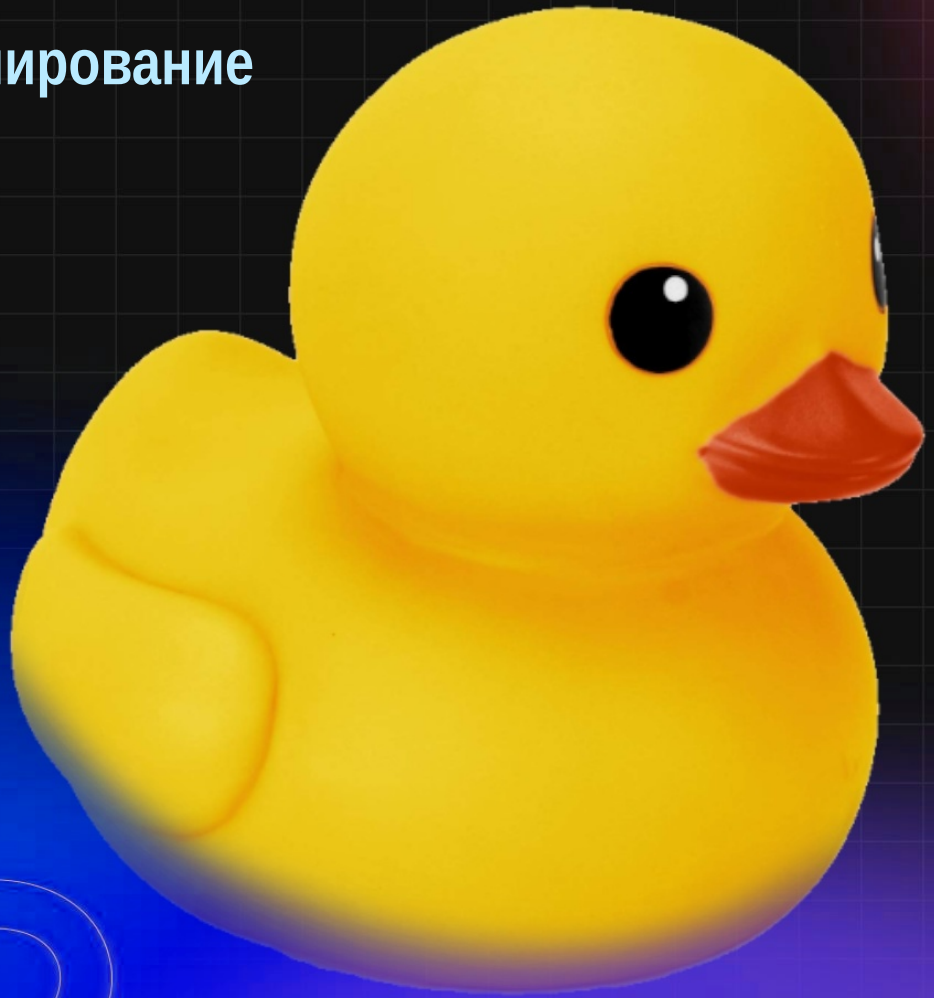
Классы-обертки (Wrappers)



Character и char

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

Программирование
2 семестр
2025



Обобщения
Generics

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

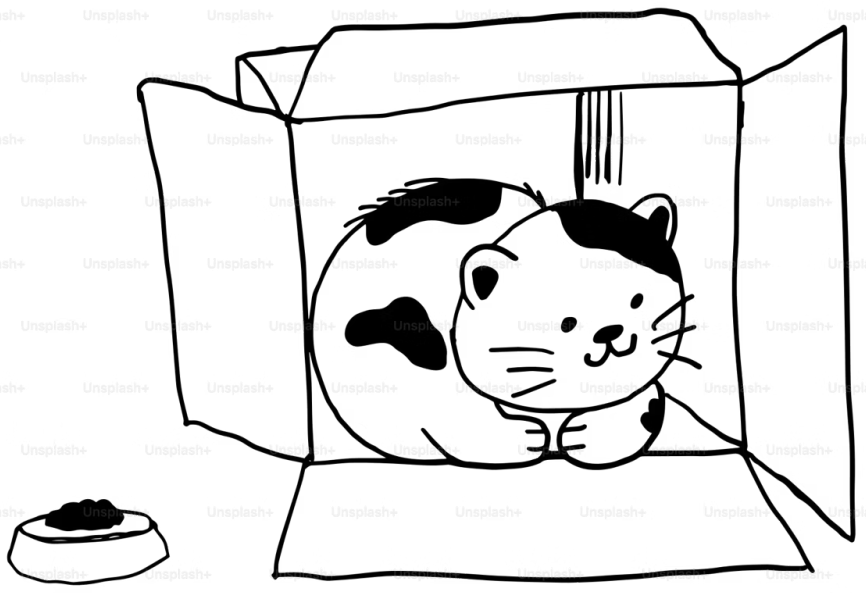
Виды полиморфизма

ИТМО

- Перегрузка методов (overriding)
  - разный код для разных типов данных
  - связывание на этапе компиляции
- Полиморфизм подтипов (ООП, overloading)
  - перегруженный код для наследуемых типов
  - связывание на этапе выполнения
- Параметричесикй полиморфизм
  - разные типы, один код
  - тип является параметром

Пример без обобщений - Box

```
public class Box {

  private Object obj;

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

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



Picture by Runend Art on Unsplash

Strging -> Box -> String ?

```
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();
```

Strging -> Box -> String ?

```
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();
```

Cannot find symbol

```
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
```

Strging -> Box -> String ?

```
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();
```

Incompatible types

```
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();
```

Strging -> Box -> String ?

```
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();

// компиляция без ошибок!
```

Integer -> Box -> String ?

```
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

```
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;
  }
}
```

String -> Box<String> -> String

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

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

```
public class Box<T> {

  private T obj;

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

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

String -> Box<String> -> String

```
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;
  }
}
```

Integer -> Box<String> = incompatible types

```
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;
  }
}
```

Generic / Parametrized

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

Обобщенный тип
(generic)

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

Параметризованный тип
(parametrized)

Рекомендуемые имена параметра типа

ИТМО

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

Java

Реализации обобщений и шаблонов

ИТМО

- 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

ИТМО

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

```
class Box<T> { }

class BBox extends Box<Byte> {

  Byte get() {
    return (Byte) super.get();
  }
}
```

```
class BBox extends Box<Byte> {
  Byte get() {
    return (Byte) super.get();
  }
  Object get() {
    return get();
  }
}
```

Мостовые методы

ИТМО

- 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);
  }
}
```

Стирание типов

ИТМО

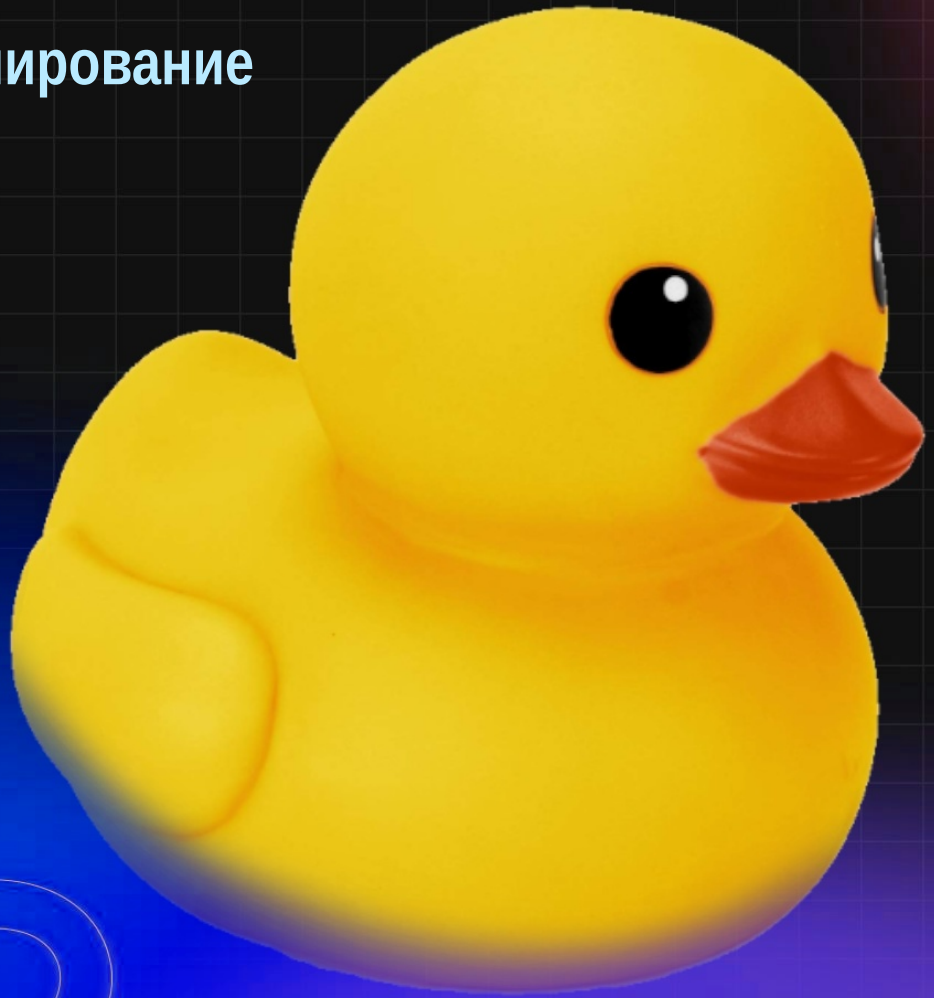
- Совместимость со старыми версиями
- Недостаточная типобезопасность
  - Контроль только при компиляции
  - При выполнении возможны нарушения

- 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>)

Java

Программирование
2 семестр
2025

ИТМО



Обобщенные методы

Обобщенные методы

- Параметр типа перед типом возвращаемого значения

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

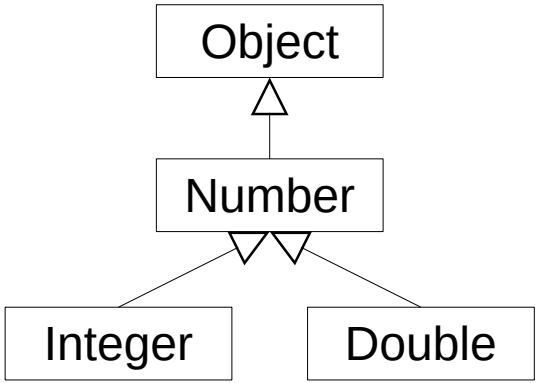
- При вызове метода нужен явный или неявный параметр

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

Ограничение типа

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

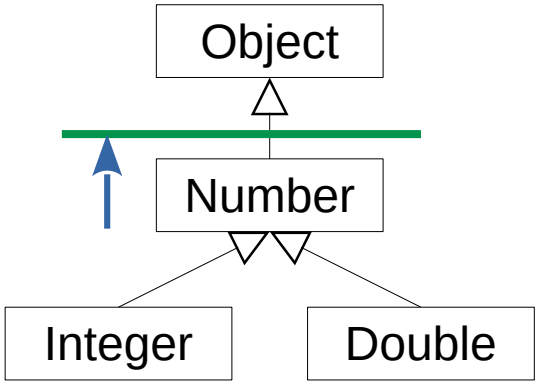
```
public class NumBox<T> {
  private Object obj

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



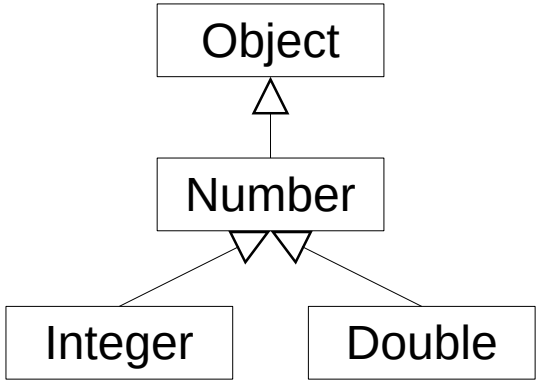
Ограничение сверху

- Инкремент только для чисел

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

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

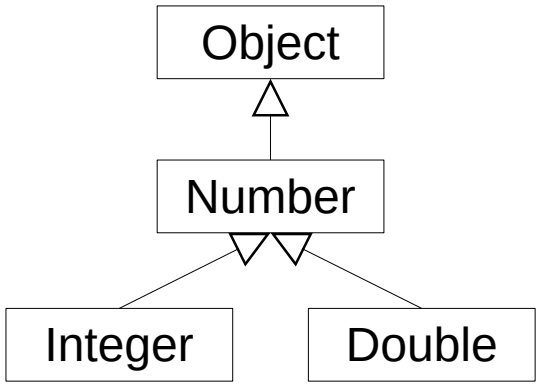


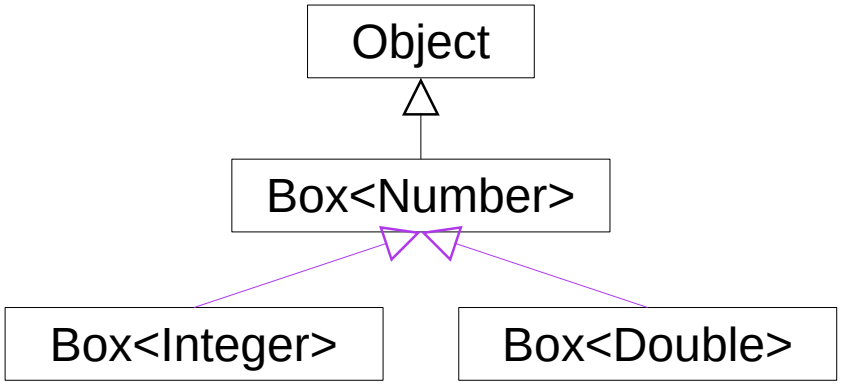
Обобщения и наследование




```
Number num;

Integer i = 42;      Double d = 3.14;
num = i;             num = d;
```

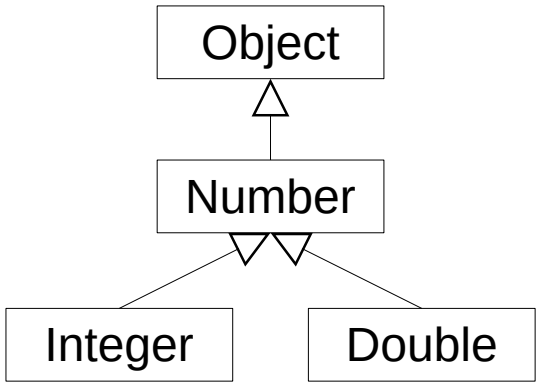
Наследование параметра типа

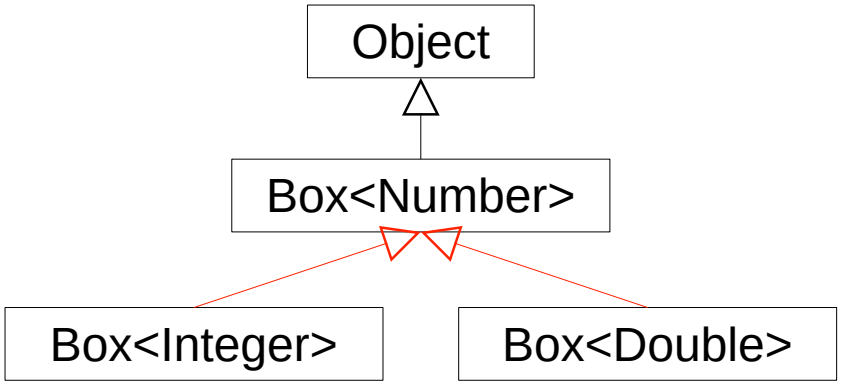





```
          Box<Number> bnum;
Integer i = 42;       Double d = 3.14;
bnum = new Box<>(i); bnum = new Box<>(d);
```

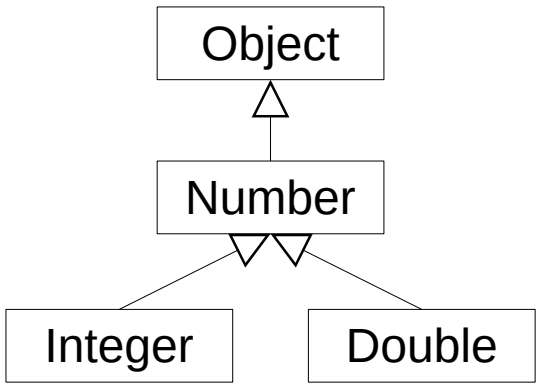
Наследование обобщений

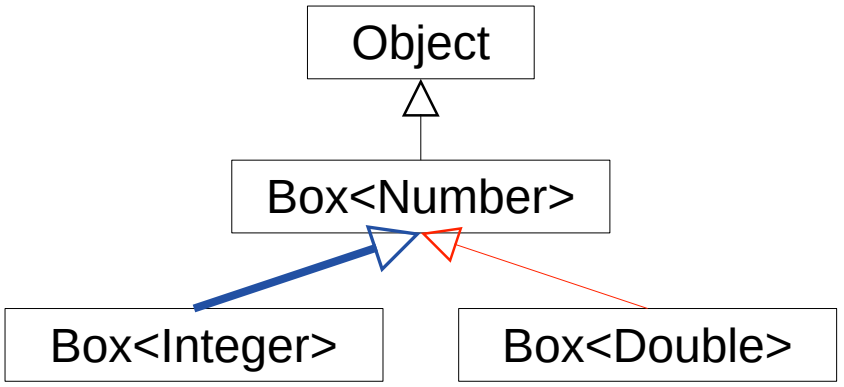





```
        Box<Integer> bint;
        Box<Number> bnum;
Integer i = 42;      Double d = 3.14;
bint = new Box<>(i);
bint = new Box<>(d);        // cannot infer type arguments for Box<>
bint = new Box<Double>(d); // Box<Double> cannot be converted to Box<Integer>
bint = new Box<Integer>(d);// Double cannot be converted to Integer
```

Тест

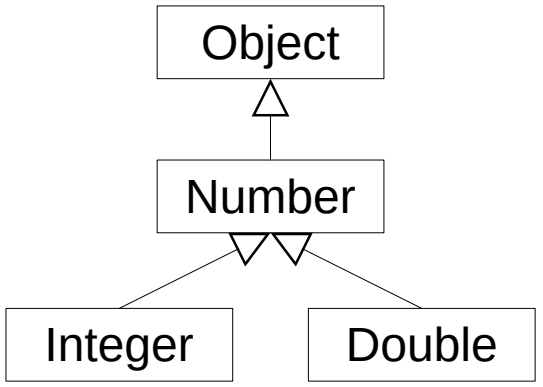


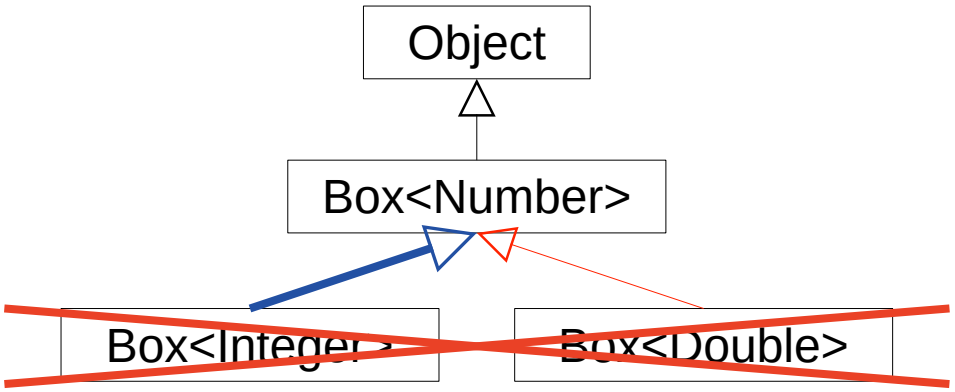



```
Box<Integer> bint;
Box<Number> bnum = (Box<Number>) bint;
```

```
Integer i = 42;       Double d = 3.14;
bnum = new Box<>(i); bnum = new Box<>(d);
```

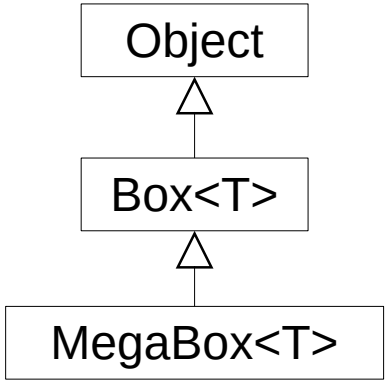
Тест

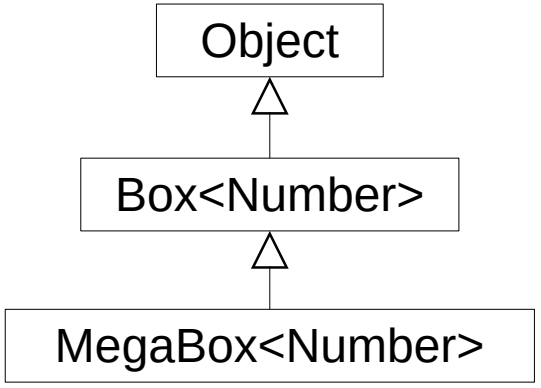





```
Box<Integer> bint;
Box<Number> bnum = (Box<Number>) bint;
                   //   Box<Integer> cannot be converted to Box<Number>
Integer i = 42;       Double d = 3.14;
bnum = new Box<>(i); bnum = new Box<>(d);
```

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 (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));
```

? - wildcard

Различие

- Box<T>
  - generic
  - T - тип
- Box<Number>
  - generic
  - тип - Number+
- Box<? extends Number>
  - generic
  - тип - Number+
- Box
  - non-generic
  - тип - any class
- Box<Object>
  - generic
  - тип - Object+ = any class
- Box<? extends Object>
- = Box<?>
  - generic
  - тип - Object+

Wildcards в Java API

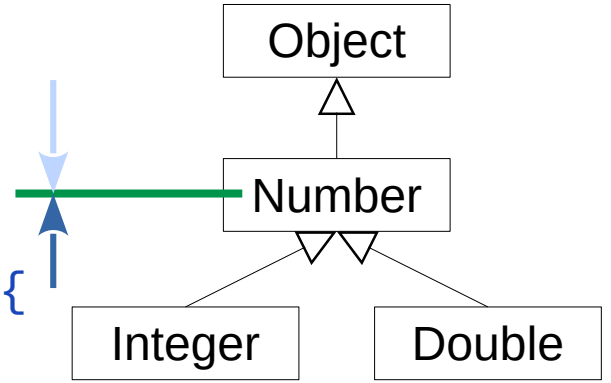
- class List<T>
  - public double sum(List<? extends Number> list)
  - public void print(List<?> list)
  - public <T> void fill(List<? super T> list, T value)
- Class java.lang.Enum<E extends Enum<E>>

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)
- 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.

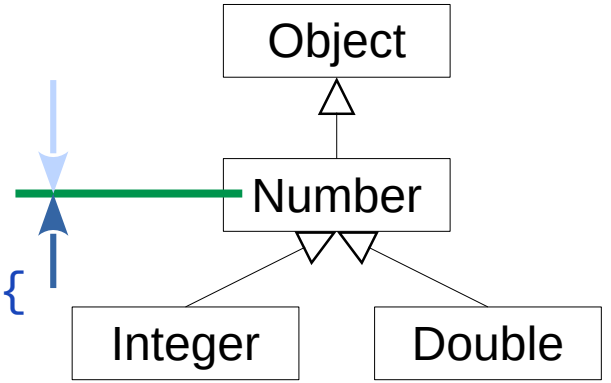
Ограничения снизу и сверху

```
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
```



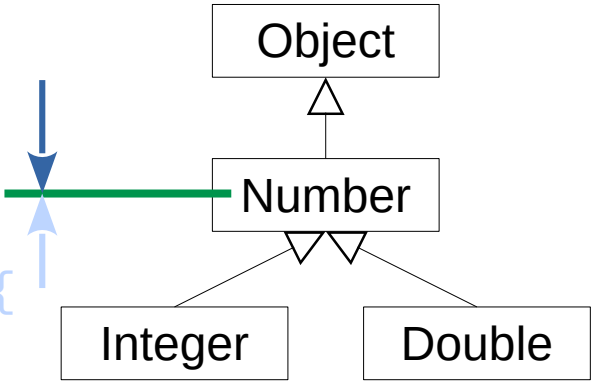
Ограничения снизу и сверху

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

  public void copyTo(Box<? super T> to) {
     to.obj = obj;
  }
  public void copyFrom(Box<? extends T> from) {
Number obj = from.obj;
  }              Integer, Double
}
Box<Number> nbox = new Box<>();
Box<Integer> ibox = new Box<>(42);
ibox.copyTo(nbox);   // T = Integer, ? = Number
nbox.copyFrom(ibox); // T = Number, ? = Integer
```



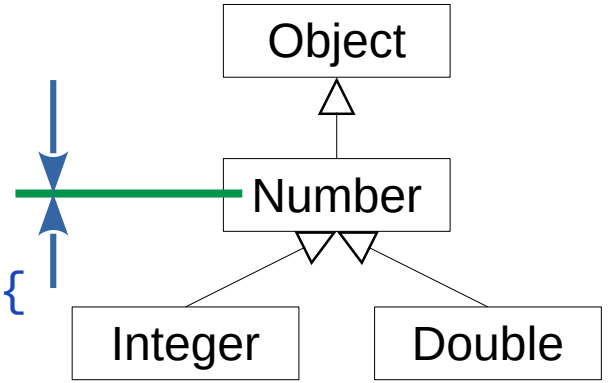
Ограничения снизу и сверху

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

  public void copyTo(Box<? super T> to) {
Number to.obj = obj;
  }              Integer, Double
  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
```



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

ИТМО

- 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)

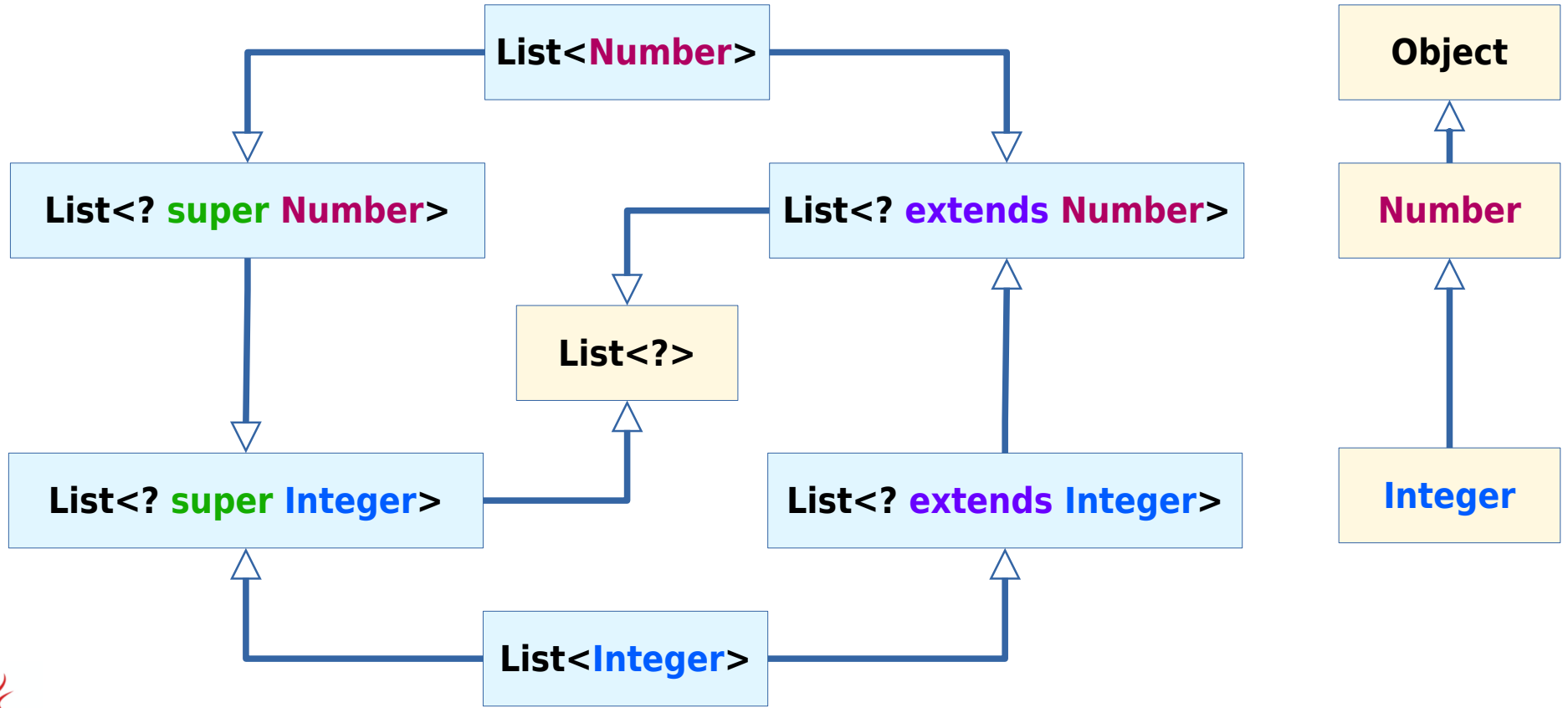
Java

Обобщения и массивы

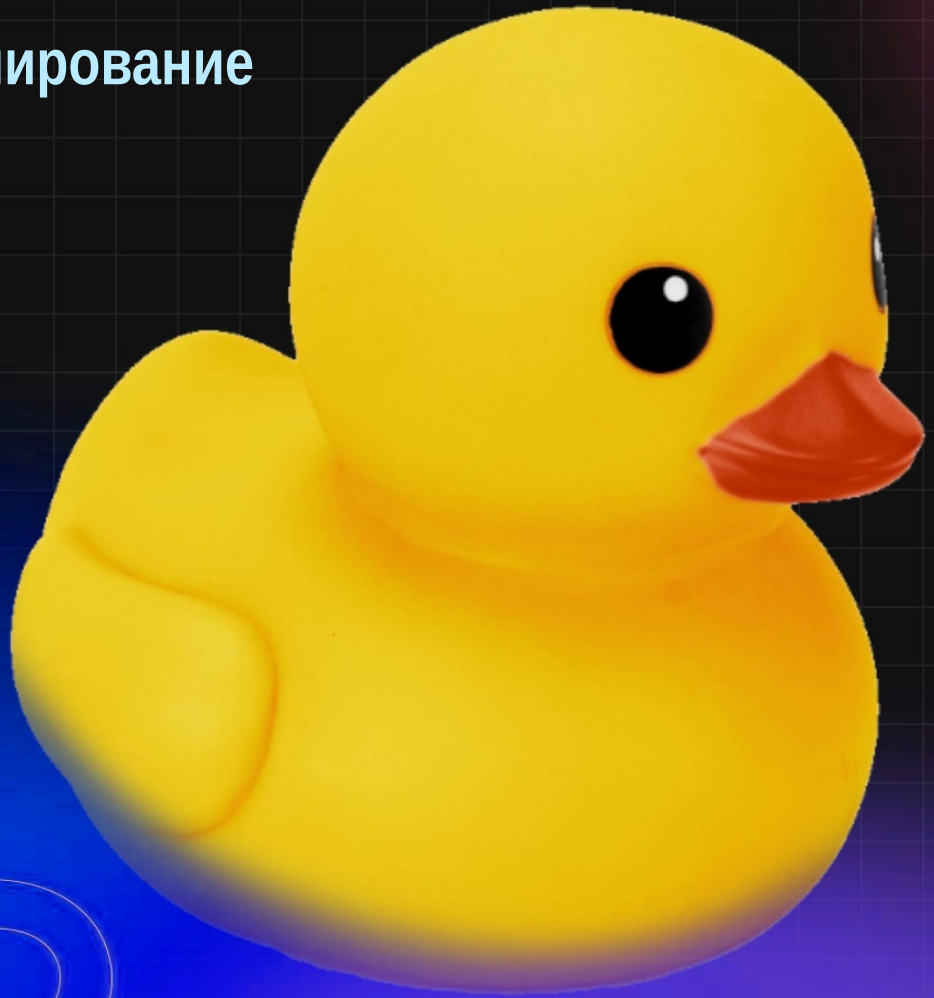
ИТМО

- 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>

Подклассы обобщенных типов



Программирование
2 семестр
2025



ИТМО

Коллекции

Массивы

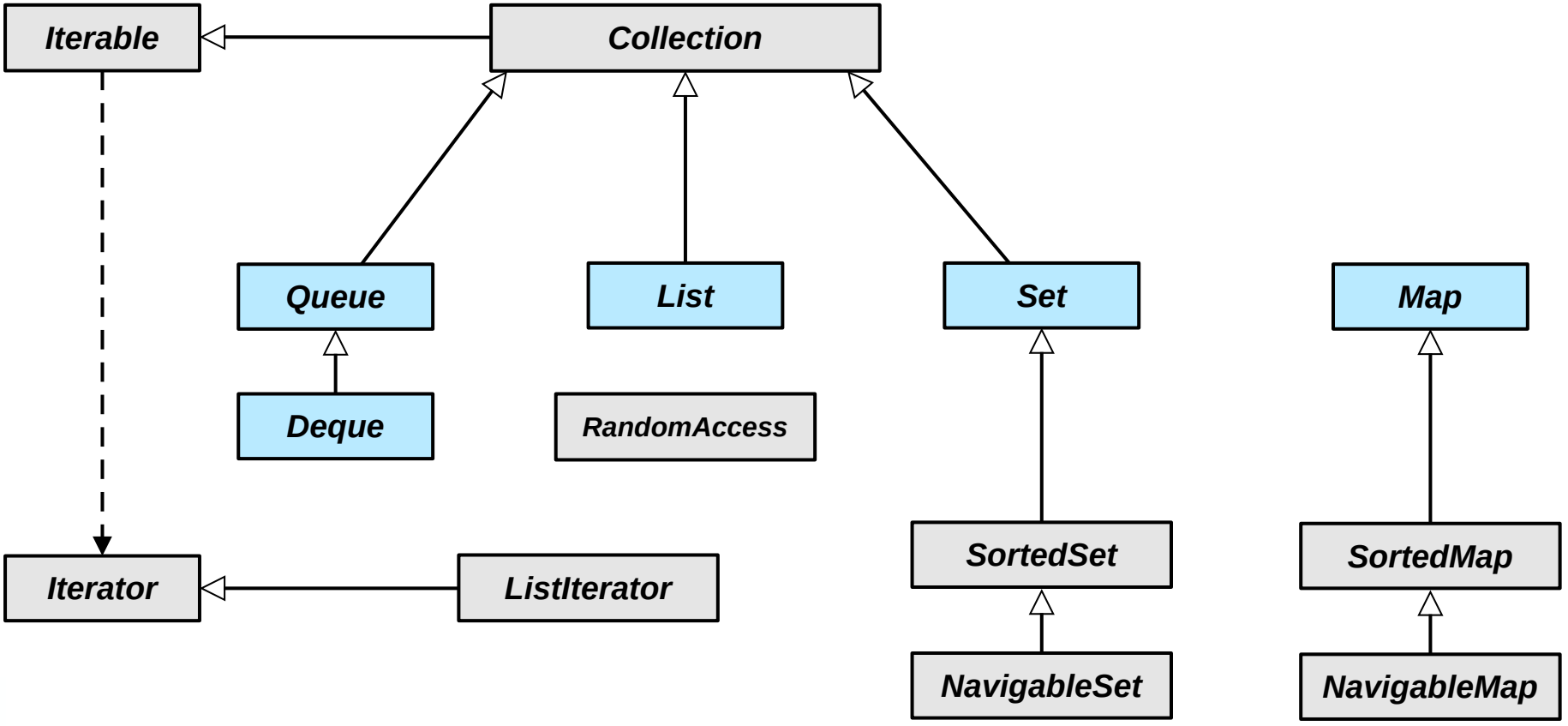
ИТМО

- 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())

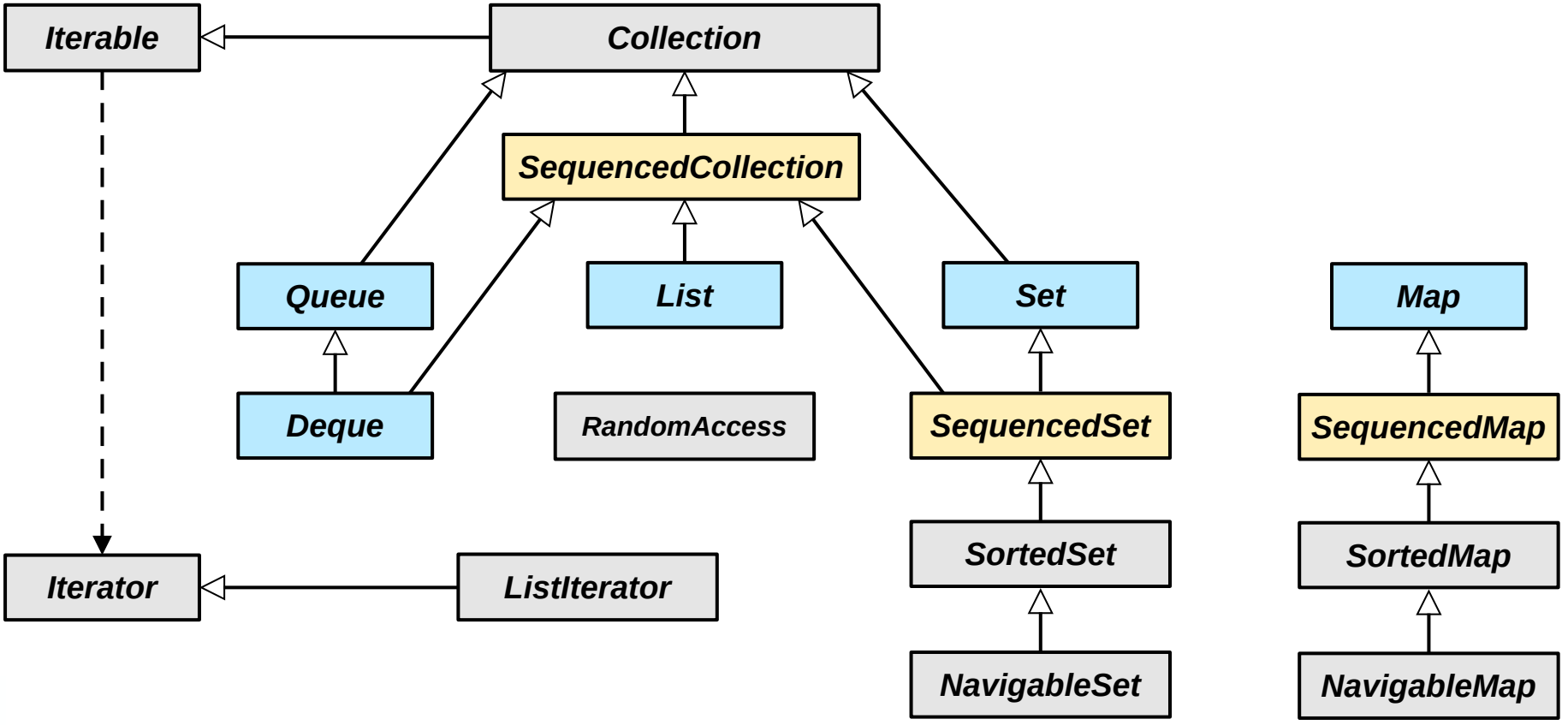
Collection Framework (java.util)

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

Основные интерфейсы



Основные интерфейсы (JDK 21)



Интерфейсы - контракты

- 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

Как это прочитать?

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

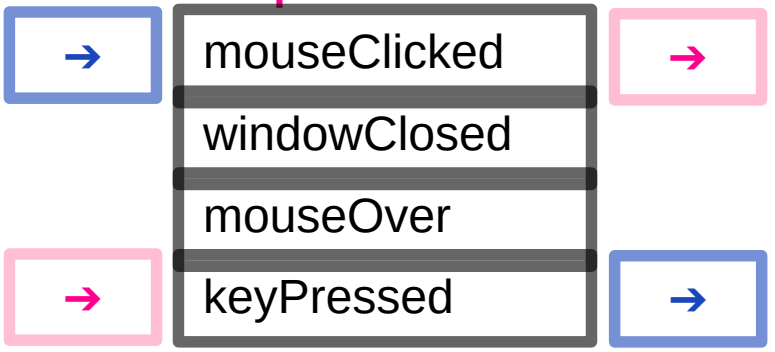
Основные категории коллекций

ИТМО

- List

0	яблоко
1	апельсин
2	банан
3	киви

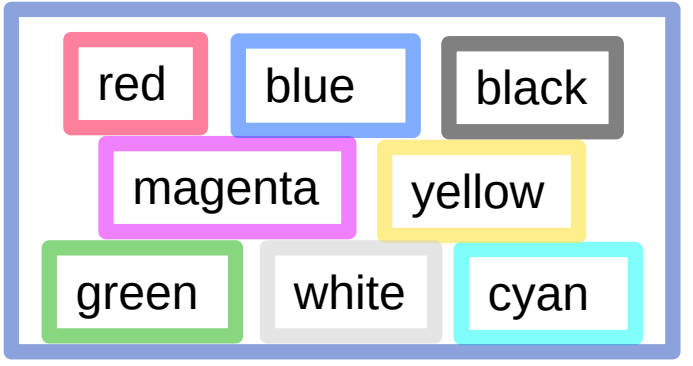
- Queue/Deque/Stack



- Map

и	37153142
т	31620970
м	16203060
о	55414481

- Set



Java

Классы - конкретные разные реализации

ИТМО

- 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)
  - $O(n^2)$
  - $O(2^n)$

Принципы

ИТМО

- 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

Java

Интерфейс Collection<E>

ИТМО

- 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

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

Внимание : тип параметра!

Массовые и другие операции

- addAll(Collection<?> c)
- removeAll(Collection<?> c)
- retainAll(Collection<?> c)
- containsAll(Collection<?> c)
- clear()
- isEmpty()
- size()

Массовые и другие операции

- addAll(Collection<?> c)
- removeAll(Collection<?> c)
- retainAll(Collection<?> c)
- containsAll(Collection<?> c)
- clear()
- isEmpty()
- size()

Внимание : тип параметра!

Интерфейс SequencedCollection<E>

- Первый (First) и последний (Last) элементы
  - void addFirst(E e), E getFirst(), E removeFirst()
  - void addLast(E e), E getLast(), E removeLast()
  - SequencedCollection reversed()

Интерфейс List<E>

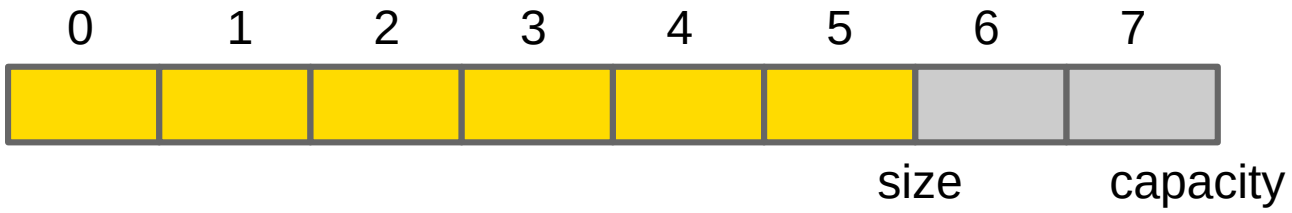
- **Индекс от 0 до N**
- E get(int index)
- E set(int index, E e)
- void add(int index. E e)
- E remove(int index)
  - List<Integer> list
  - list.remove(1) ???

- 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();
```

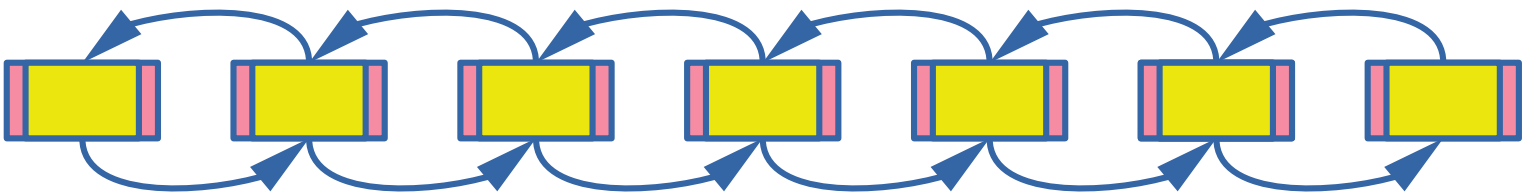
Динамический массив ArrayList

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



Двусвязный список LinkedList

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



Еще списки

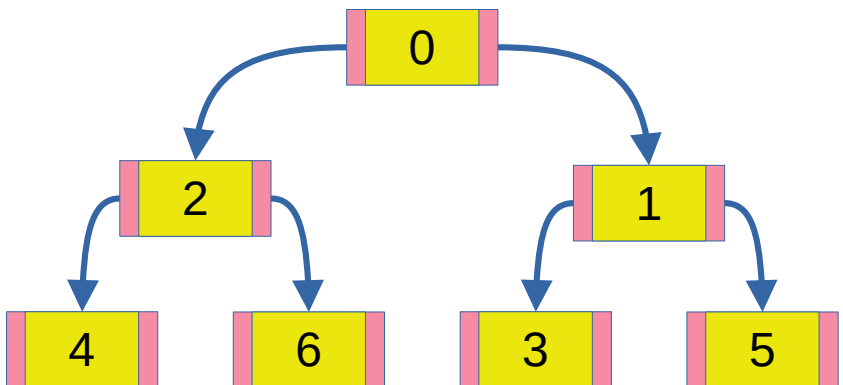
- Vector
  - потокобезопасный, но устаревший ArrayList
- Stack
  - устаревшая реализация на базе вектора
- CopyOnWriteArrayList
  - новая копия при каждом изменении

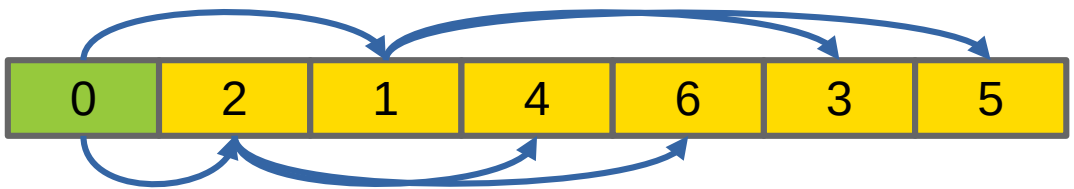
Интерфейс Queue<E>

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

Приоритетная очередь PriorityQueue

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





Интерфейс Deque<E>

ИТМО

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

- Пример

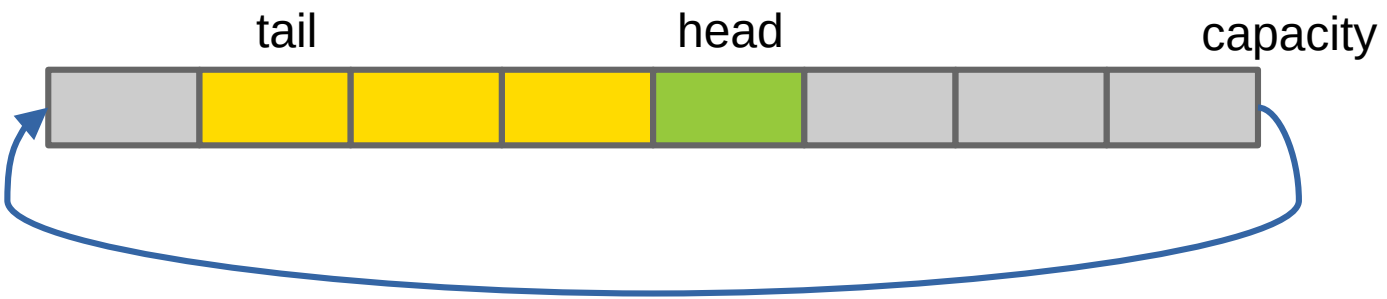
```
Queue<String> queue = new ArrayDeque<>();
queue.add("First element, index 0");
queue.add("One more");
var list = List<String>.of("X", "Y");
queue.add(list);
while (!queue.empty()) {
    System.out.println(queue.remove());
}
```

Deque<E> как Queue и Stack

Queue	Deque	Stack
	addFirst(e)	push(e)
add(e)	addLast(e)	
offer(e)	offerLast(e)	
remove()	removeFirst()	pop()
poll()	pollFirst()	
element()	getFirst()	peek()
peek()	peekFirst()	

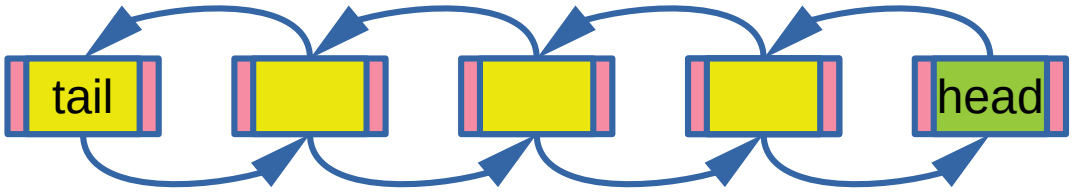
Дек на основе массива - ArrayDeque

- 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>

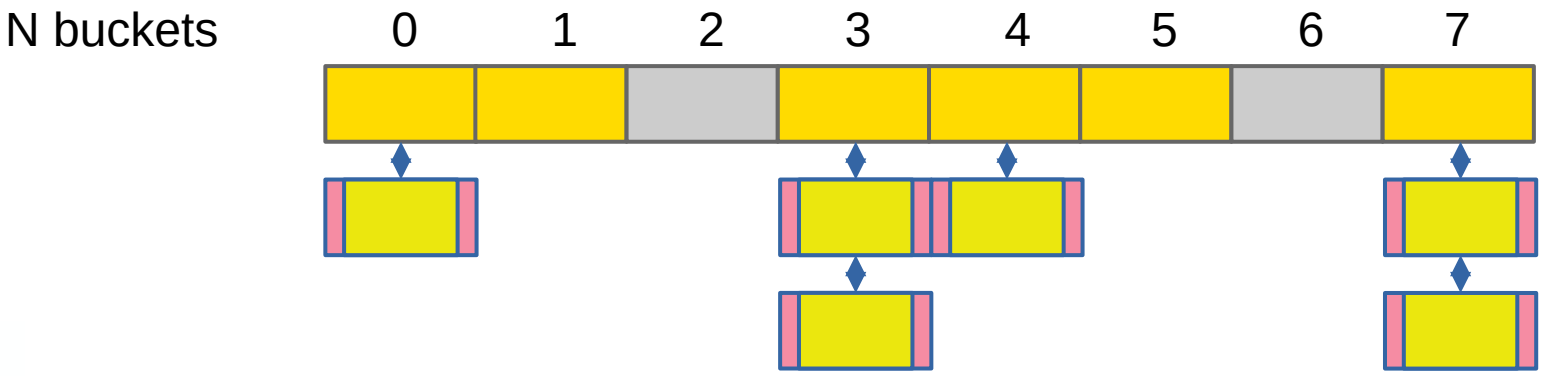
- **Ассоциативный массив, отображение, словарь**
- Два параметра типа
- Ключи уникальные, значения - любые
- Методы:
  - V put(K key, V value)
  - V get(Object K)
  - V remove(Object k)
- Представления:
  - keySet(), values(), entrySet()

- Пример

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

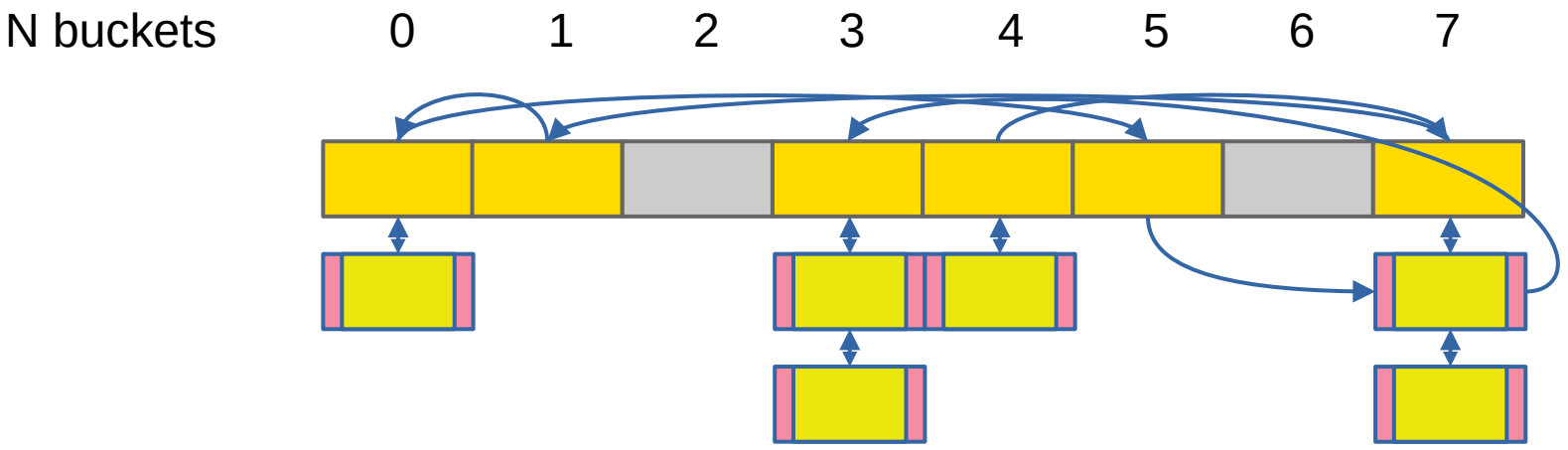
Хеш-таблица HashMap

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



SequencedMap + LinkedHashMap

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



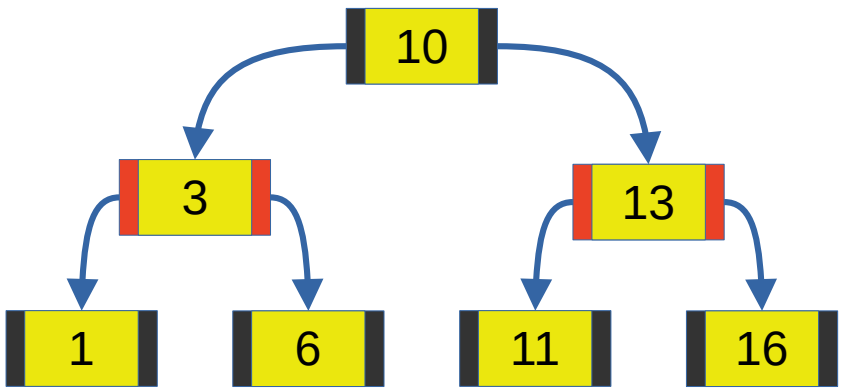
Интерфейсы SortedMap, NavigableMap

ИТМО

- 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)

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
  - сравнение ключей по ==, а не по equals()

Интерфейс Set<E>

ИТМО

- 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

Еще множества

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

Методы of - неизменяемый List, Set, Map

- 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)

класс Collections

ИТМО

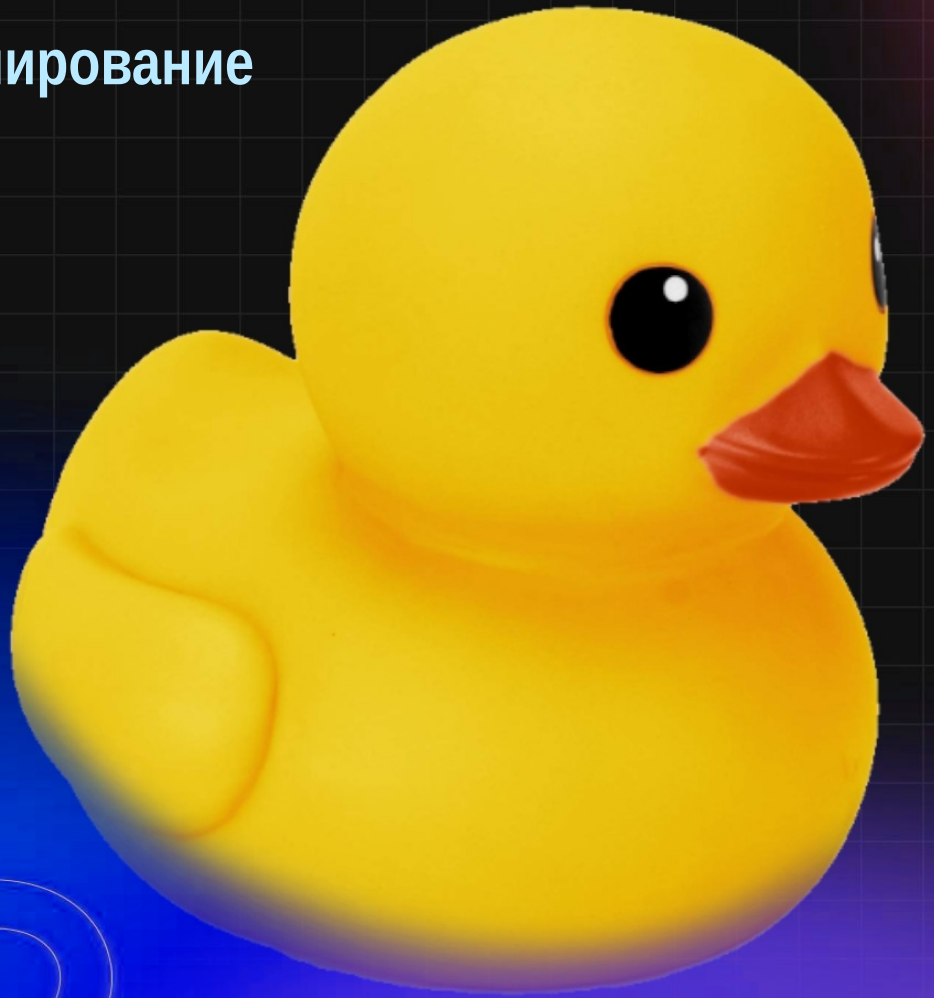
- коллекции-обертки (представления):
  - synchronized
  - unmodifiable
  - checked
- алгоритмы для List
  - sort(), shuffle(), reverse(), fill(), swap(), binarySearch()
- фабрики коллекций
  - emptySet, emptyList, emptyMap,
  - singleton, singletonList, singletonMap
- и еще много полезного (Queue asLifoQueue(Deque))

Java

Класс Arrays

- 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

Программирование
2 семестр
2025



ИТМО

Шаблоны проектирования (введение)

Шаблоны проектирования

- Кто-то когда-то уже делал что-то похожее
- Пришлось вносить изменения - возникли проблемы
- Нужно сразу было делать по-другому!

- GoF Book
  - Gang of Four

Основные принципы проектирования

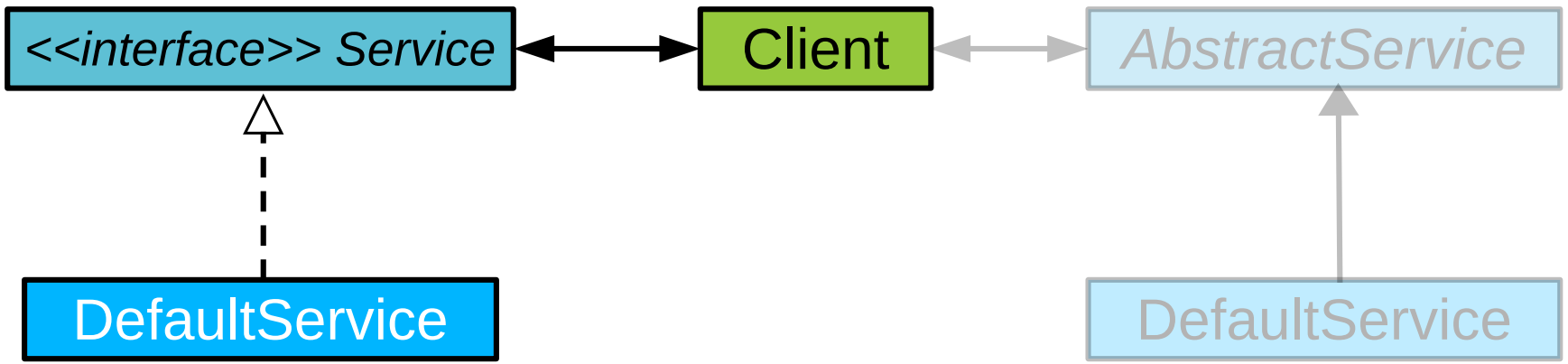
ИТМО

- Повторное использование кода
  - DRY - Don't repeat yourself
  - стандартные библиотеки
  - фреймворки
- Расширяемость
  - Учет возможных будущих изменений

Java

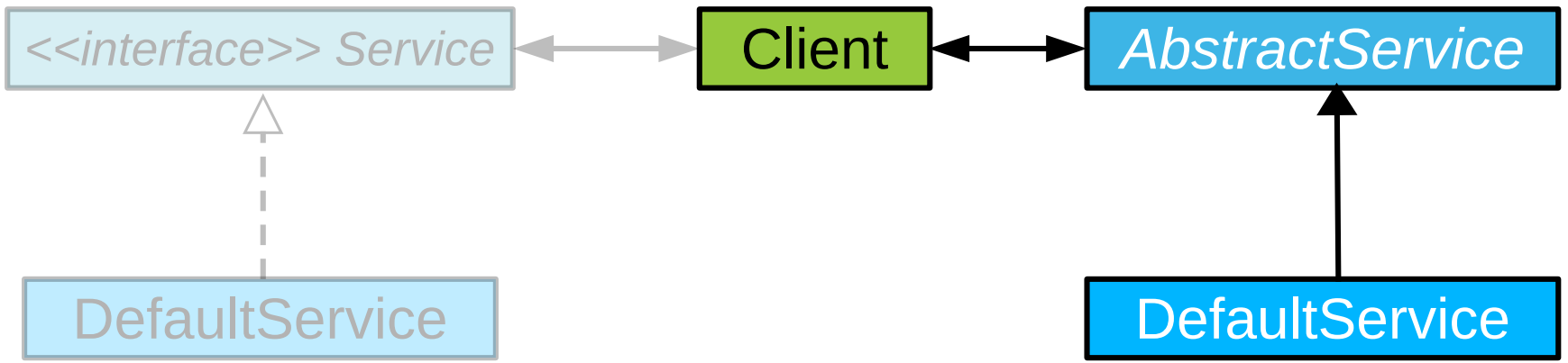
Интерфейс / Абстрактный суперкласс

- Независимость от реализации - больше гибкость и универсальность
- Унификация поведения потомков - проще реализация



Интерфейс / Абстрактный суперкласс

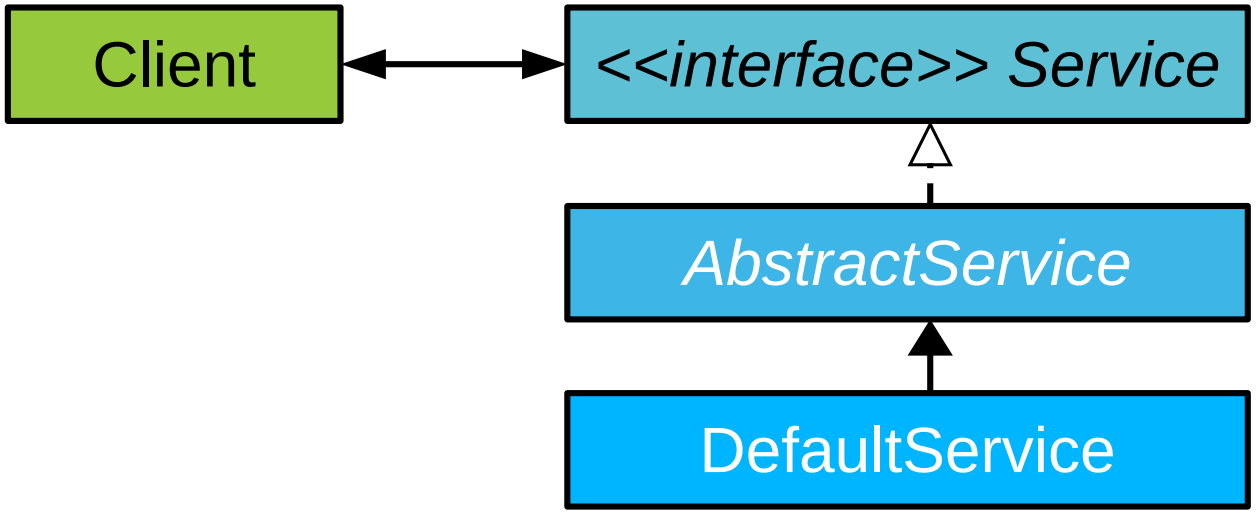
- Независимость от реализации - больше гибкость и универсальность
- Унификация поведения потомков - проще реализация



Интерфейс + абстрактный суперкласс

ИТМО

- Общие свойства - АБСТРАКТНЫЙ КЛАСС
- Взаимодействие с клиентом - ИНТЕРФЕЙС
- Set<T> ⇠ AbstractSet<T> ⇦ HashSet<T>



Java

Наследование и делегирование

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)

```
class Dog extends Animal  { }
```

Java

Наследование и делегирование

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)
- Student и Person
  - студент - **роль**

```
class Dog extends Animal  { }

class Person { }

class Student extends Person { }
class Teacher extends Person { }
```

Java

Наследование и делегирование

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)
- Student и Person
  - студент - **роль**
  - роль может меняться
- Делегирование!

```
class Dog extends Animal  { }

class Person {
  public getName() { }
}

class Student {
  Person p;
  public getName() {
    return p.getName();
  }
}
```

Наследование и композиция

ИТМО

- Наследование - подкласс
  - собака - подкласс животного
  - статическое отношение **is-a** (Dog is an animal)
- Композиция
  - "часть-целое"
  - отношение **has-a**

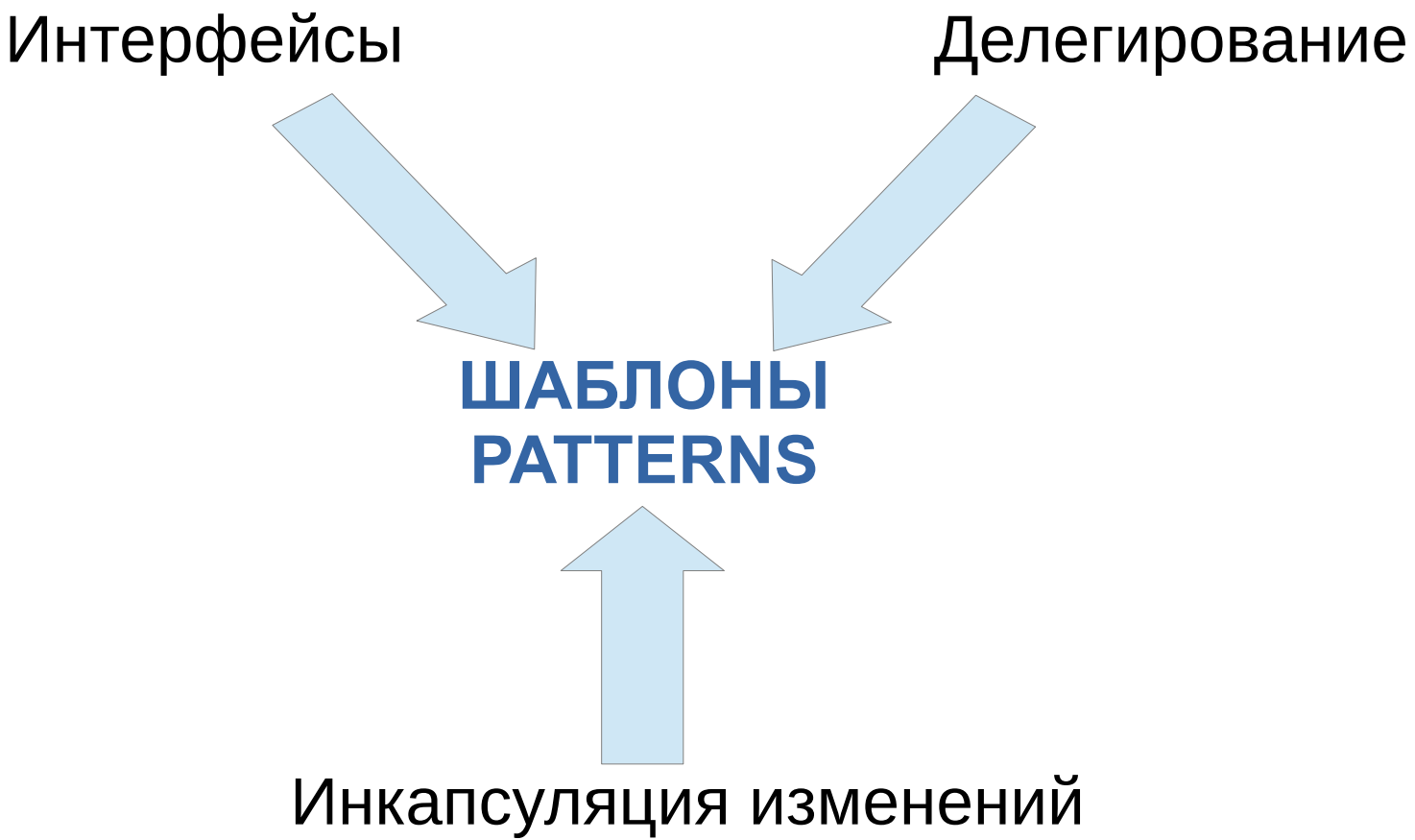
```
class Dog extends Animal  { }

class Car {
  Engine engine;
  Wheel[] wheels;
  Door[] doors;
}
```

Java

Инкапсуляция изменений

- Отделить изменяющееся от постоянного
- Инкапсулировать изменяющееся

Как справиться с изменениями?

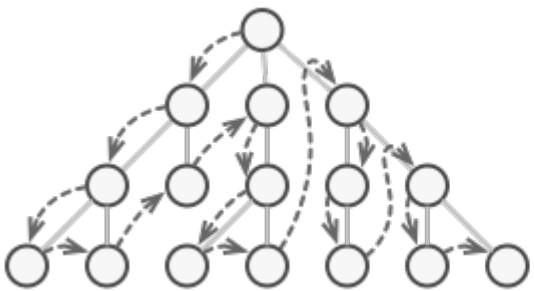


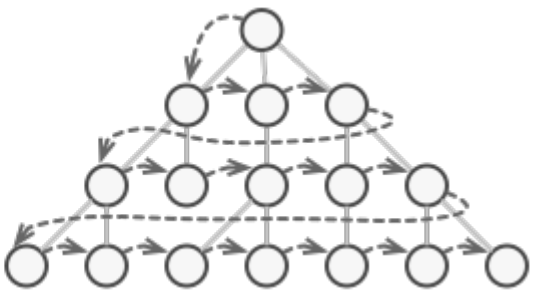
Виды шаблонов

- Порождающие
  - Singleton, Factory Method, Builder, ...
- Структурные
  - Adapter, Decorator, Proxy
- Поведенческие
  - Command, Iterator, Observer, Strategy

Iterator

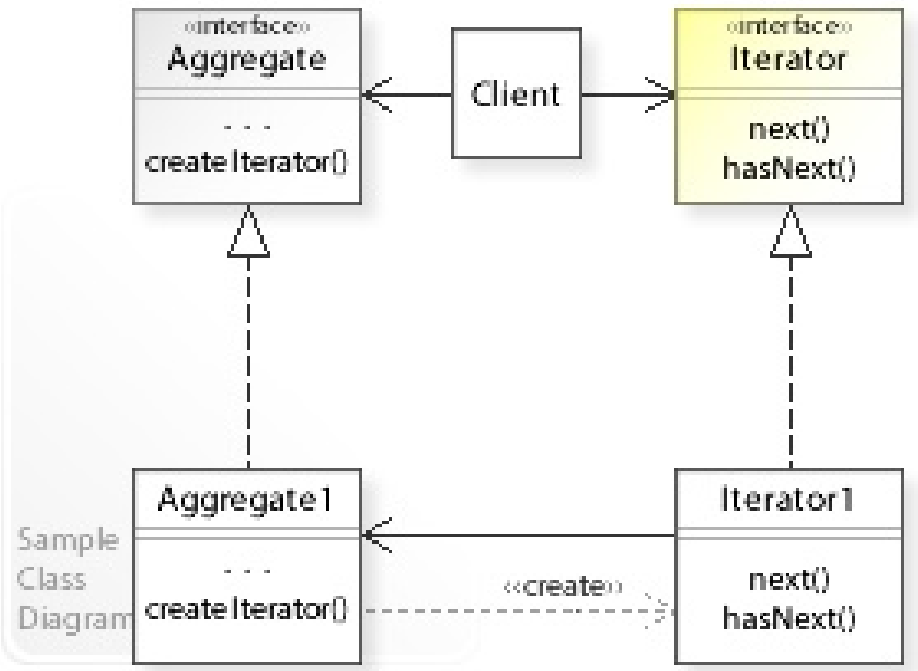
- Последовательный доступ к элементам коллекции
- Итератор знает внутреннюю структуру коллекции
- Разные способы обхода





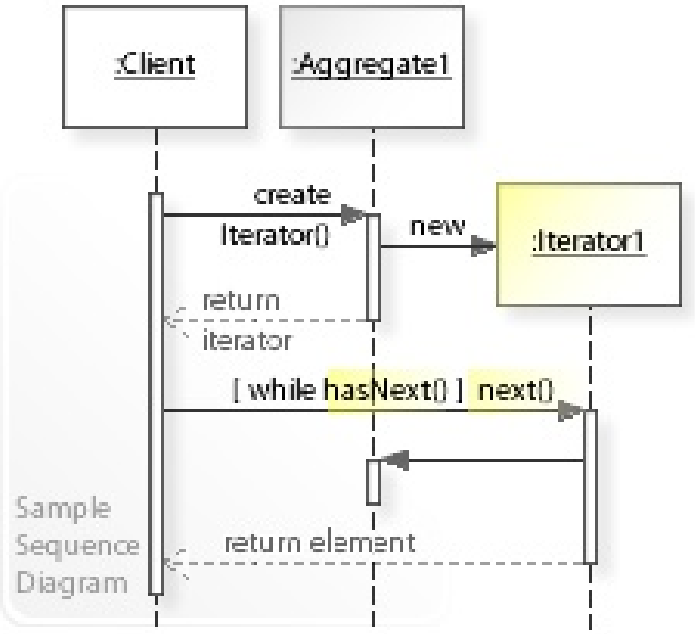
Iterator

- Последовательный доступ к элементам коллекции
- Экскурсия
  - Реальный гид
  - Аудиогид
  - Путеводитель
- Java
  - java.util.Iterator
  - java.util.Scanner



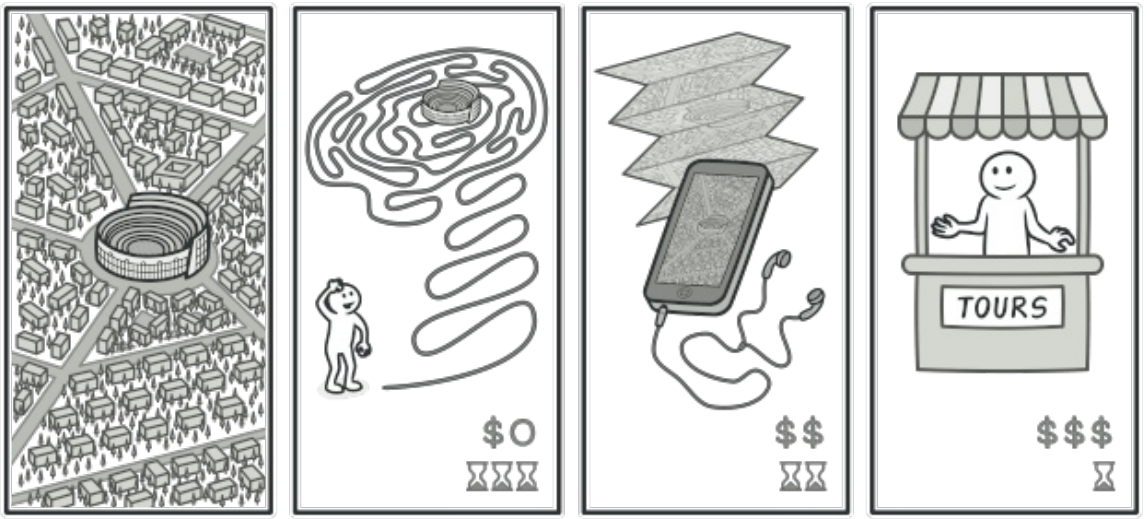
Iterator

- Последовательный доступ к элементам коллекции
- Клиент запрашивает итератор у структуры
- Структура возвращает свой итератор клиенту
- Клиент запрашивает элементы у итератора по очереди



Iterator

- Универсальный доступ ко всем коллекциям
- Гибкая реализация обхода структур
- Более сложный вариант, чем простой цикл



Интерфейс java.lang.Iterable<T>

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

Интерфейс Iterator<E>

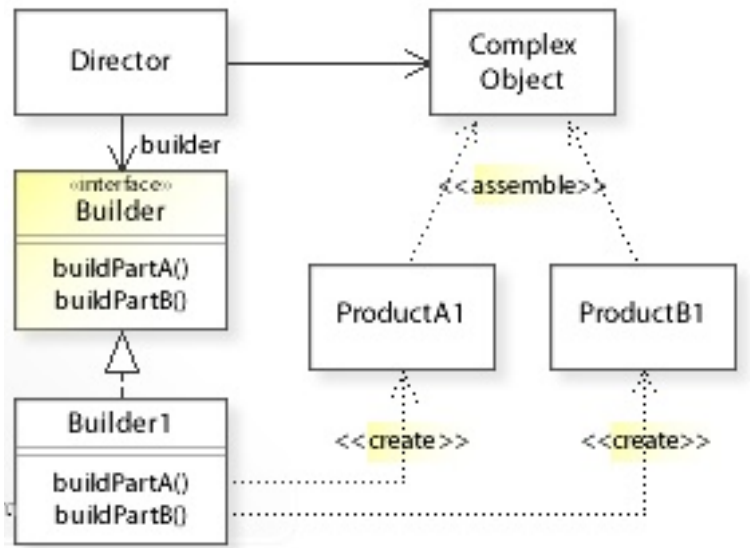
- **Реализует обход** элементов коллекции
- Методы:
  - boolean hasNext()
  - E next()
  - void remove()

- Пример

```
Iterator<Short> it = c.iterator();
while (it.hasNext()) {
  Short s = it.next();
  System.out.println(s);
  if (s == 0) {
    it.remove();
  }
}
```

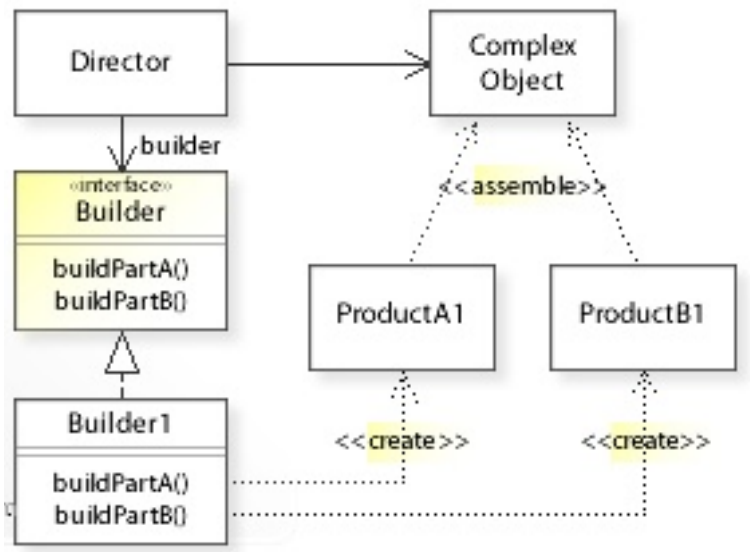
Builder

- Строитель
  - Строит сложный объект по шагам



Builder

- Интерфейс Builder объявляет этапы создания продуктов
- Конкретные строители реализуют этапы создания
- Продукт — создается строителем при вызове метода create()
- Директор управляет строителем для производства нужной конфигурации продукта



Builder

- Класс со множеством полей
- При создании какие-то поля нужно задать
- Возможно не все

```
class Trip {
  private Location[] locations;
  private Transport transport;
  private int numberOfPersons;
  private Date startDate;
  private Date finalDate;
  private Hotel hotel;
  ...
}
```

Builder

- Телескопические конструкторы - ужас №1

```
class Trip {
  public Trip(Date date)
  public Trip(Location location)
  public Trip(Date date, Location location)
  public Trip(Date date, Transport transport)
  public Trip(Date date, Location location, Transport transport)
  public Trip(Date date, Location location, int guests)
  public Trip(Date from, Date until, Location localtion)
  ...
}
```

Builder

- Аргументы по умолчанию - ужас №2

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,),
activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
max_iter=200, shuffle=True, random_state=None, tol=0.0001,
verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9,
beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

Builder

- Простой конструктор и набор сеттеров
- Объект не полностью готов в промежуточном состоянии

```
class Trip {
  public Trip()
  public setLocation(Location)
  public setDate(Date)
  public setTransport(Transport)
  ...
}
```

Builder

- Строитель создает объект постепенно
- Методы возвращают строителя
- В конце создаем готовый объект

```
Builder b = new Builder()
   .setWhere("Moscow")
   .setTransport("train")
   .setDate(1,7,2023);
   ...

Trip trip = b.createTrip();
```

- StringBuilder .append()

Builder

- Строитель создает объект постепенно
- Методы возвращают строителя
- В конце создаем готовый объект

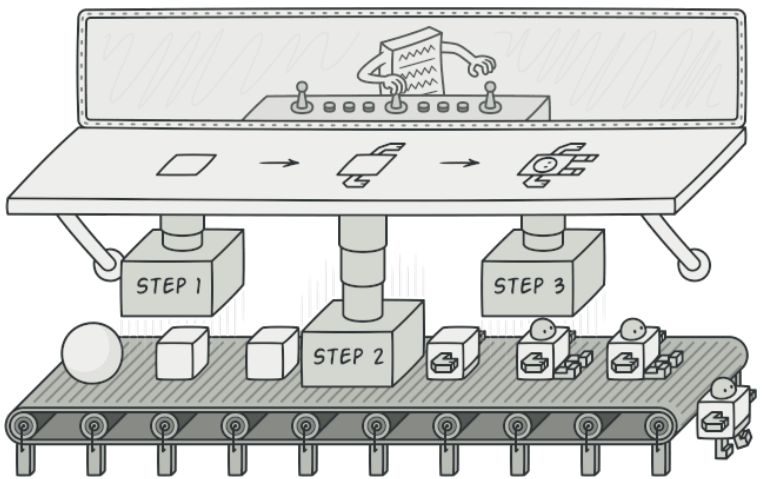
```
Builder b = new Builder()
   .setWhere("Moscow")
   .setTransport("train")
   .setDate(1,7,2023);
   ...

Trip trip = b.createTrip();
```

- Директор создает объект с помощью строителя одним методом

```
Director d = new Director();
Trip trip = d.makeTrainTrip("Moscow", new Date(1,7,2023));
```

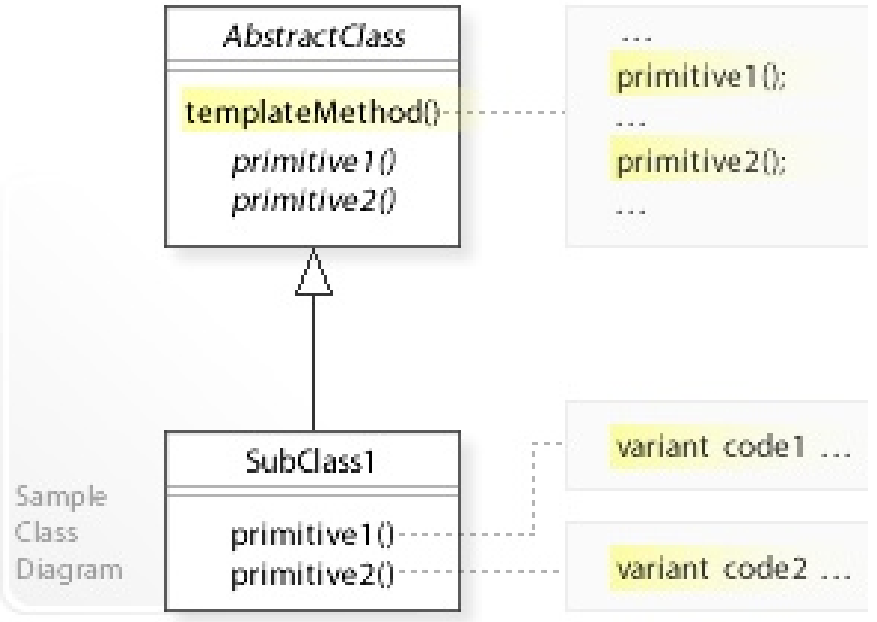
Builder

- Изоляция сложной сборки от бизнес-логики
- Разрешает пошаговое создание
- Усложнение структуры программы (+ строитель)



Шаблон Template Method

- Позволяет реализовать часть поведения в базовом классе, остальное реализуется в подклассах
- Шаги
  - абстрактные
  - дефолтные
  - хуки (изначально пустые)
- Покемоны
  - Move.applySelfDamage()



Template Method

- Упрощает повторное использование кода
- Жестко задает последовательность действий
- При большом количестве действий сложно поддерживать