

# КРОК

## Java: Generics & Collections Framework

Александр Головин

Ведущий инженер-разработчик



# Тестирование, выполняемое разработчиками

Тестирование - самая популярная методика повышения качества, подкрепленная многими исследованиями и богатым опытом разработки коммерческих приложений.

## Виды тестирования:

- Модульные
- Интеграционные
- Нагрузочные
- ...

Зачем нам тесты?

Test-Driven Development(TDD)

# JUnit: Модульные тесты

Тест - программа, которая проверяет корректность работы другой программы.

Тесты JUnit представляет собой класс, методы которого помечены аннотациями:

- **@BeforeAll** - до всех(static метод)
- **@BeforeEach** - перед каждым тестом
- **@Test** - тест
- **@AfterEach** - после каждого теста
- **@AfterAll** - после всех(static метод)



# JUnit: Проверки

```
public class Calc {  
    public int sum(int a, int b) {  
        return a + b;  
    }  
}
```

```
public class CalcTest {  
    @Test  
    public void testSum() {  
        final Calc calc = new Calc();  
        Assertions.assertEquals(2, calc.sum(1, 1));  
        Assertions.assertEquals(0, calc.sum(-1, 1));  
        Assertions.assertTrue(  
            Integer.MAX_VALUE > calc.sum(Integer.MAX_VALUE, 1)  
        );  
    }  
}
```

# Как добавить тесты в проект

- Подключить библиотеку JUnit 5
- Создать директорию test для исходников тестов
- Создать класс <имя тестируемого класса>Test для тестируемого класса

## Задача. Реализовать Cache.

```
public class User {  
    private String login;  
    private String name;  
    ...  
}
```

- Получать объект User долго
- Одного и того же пользователя требуется получать неоднократно
- Нужно ускорить процесс получения



## equals & hashCode

- Если во время работы приложения несколько раз обратиться к одному и тому же объекту, метод `hashCode` должен постоянно возвращать одно и то же целое число
- Если метод `equals(Object)` показывает, что два объекта равны друг другу, то, вызвав для каждого из них метод `hashCode`, вы должны получить в обоих случаях одно и то же целое число
- Если метод `equals(Object)` показывает, что два объекта друг другу не равны, вовсе не обязательно, что метод `hashCode` возвратит для них разные числа

# Wrapper Classes and Autoboxing

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double



## Generic - это просто -)

```
elif operation == "MIRROR_X":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_o
```

```
public static <T, K, U, M extends Map<K, U>>
Collector<T, ?, M> toMap(Function<? super T, ? extends K> keyMapper,
                        Function<? super T, ? extends U> valueMapper,
                        BinaryOperator<U> mergeFunction,
                        Supplier<M> mapFactory) {...}
```

# Diamond Operator

- `List names = new ArrayList();`
- `List<String> names = new ArrayList<String>();`
- `List<String> names = new ArrayList<>();`
  
- `Map<String, HashMap<String, String>> map = new HashMap<String, HashMap<String, String>>();`
- `Map<String, HashMap<String, String>> map = new HashMap<>();`

# Java Generic. Why?

```
List languages = new ArrayList();  
languages.add("Java");
```

```
for (String language : languages) {  
    System.out.println(language);  
}
```

+ `ClassCastException`



# Generic Classes

```
public class Optional<T> {  
    private T value;  
  
    public Optional(T value) {  
        this.value = value;  
    }  
  
    public boolean isPresent() {  
        return value != null;  
    }  
  
    public T get() {  
        return value;  
    }  
}
```

# Generic Methods

```
public class Optional<T> {  
    private T value;  
  
    private Optional(T value) {  
        this.value = value;  
    }  
  
    private static <T> Optional<T> create(T value) {  
        return new Optional<>(value);  
    }  
}
```

# Naming Conventions for Generics


E - Element

N - Number

T - Type

V - Value

S,U,V etc. - 2nd, 3rd, 4th types



```
elif operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o  
    #mirror_ob.select = 0  
time = bpy.context.selected  
time = bpy.context.selected
```



# Restrictions on Generics

- Cannot Instantiate Generic Types with Primitive Types
- Cannot Create Instances of Type Parameters
- Cannot Declare Static Fields Whose Types are Type Parameters
- Cannot Use Casts or instanceof With Parameterized Types
- Cannot Create Arrays of Parameterized Types
- Cannot Create, Catch, or Throw Objects of Parameterized Types
- Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type

## Задача. Generic Cache.

Появились новые классы. И для них тоже нужно реализовать Cache.

time to coding ...

# Generic Upper Bounds

```
elif operation == "MIRROR_X":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_ob))
#mirror_ob.select = 0
#bpy.context.scene.objects.active = mirror_ob
#bpy.context.scene.objects.active = modifier_ob
```

```
Optional<? extends B> v1 = Optional.create(new A());
Optional<? extends B> v2 = Optional.create(new B());
Optional<? extends B> v3 = Optional.create(new C());
Object o = v2.getValue();
A a = v2.getValue();
B b = v2.getValue();
C c = v3.getValue();
```



# Generic Lower Bounds

```
elif operation == "MIRROR_X":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_ob))
#mirror_ob.select = 0
#bpy.context.scene.objects.active = mirror_ob
#bpy.context.scene.objects.active = modifier_ob
```

```
Optional<? super B> v1 = Optional.create(new A());
Optional<? super B> v2 = Optional.create(new B());
Optional<? super B> v3 = Optional.create(new C());
Object o = v2.getValue();
A a = v2.getValue();
B b = v2.getValue();
C c = v3.getValue();
```

## Вернемся к началу ...

```
elif operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```

```
public static <T, K, U, M extends Map<K, U>>  
Collector<T, ?, M> toMap(Function<? super T, ? extends K> keyMapper,  
                        Function<? super T, ? extends U> valueMapper,  
                        BinaryOperator<U> mergeFunction,  
                        Supplier<M> mapFactory) {...}
```

# Comparator vs Comparable

```
public interface Comparator<T> {  
    /** Compares its two arguments for order.  Returns a negative integer, ...*/  
    @Contract(pure = true)  
    int compare(T o1, T o2);  
  
    /** Indicates whether some other object is &quot;equal to&quot; this ...*/  
    boolean equals(Object obj);  
}
```

```
public interface Comparable<T> {  
    /** Compares this object with the specified object for order.  Returns a ...*/  
    @Contract(pure = true)  
    public int compareTo(@NotNull T o);  
}
```



# Java Collection Framework

```
elif_operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif_operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```

■ ■ ■

# Iterable & Iterator

```
elif operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```

```
I 🔒 Iterable  
  (m) 🔒 iterator(): Iterator<T>  
  (m) 🔒 forEach(Consumer<? super T>): void  
  (m) 🔒 spliterator(): Spliterator<T>
```

```
I 🔒 Iterator  
  (m) 🔒 hasNext(): boolean  
  (m) 🔒 next(): E  
  (m) 🔒 remove(): void  
  (m) 🔒 forEachRemaining(Consumer<? super E>): void
```

# Collection

```
elif operation == "MIRROR_X":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True
```

```
#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_o
```

## Collection

▶ Iterable

▶ Object

size(): int

isEmpty(): boolean

contains(Object): boolean

toArray(): Object[]

toArray(T[]): T[]

toArray(IntFunction<T[]>): T[]

add(E): boolean

remove(Object): boolean

containsAll(Collection<?>): boolean

addAll(Collection<? extends E>): boolean

removeAll(Collection<?>): boolean

removeIf(Predicate<? super E>): boolean

retainAll(Collection<?>): boolean

clear(): void

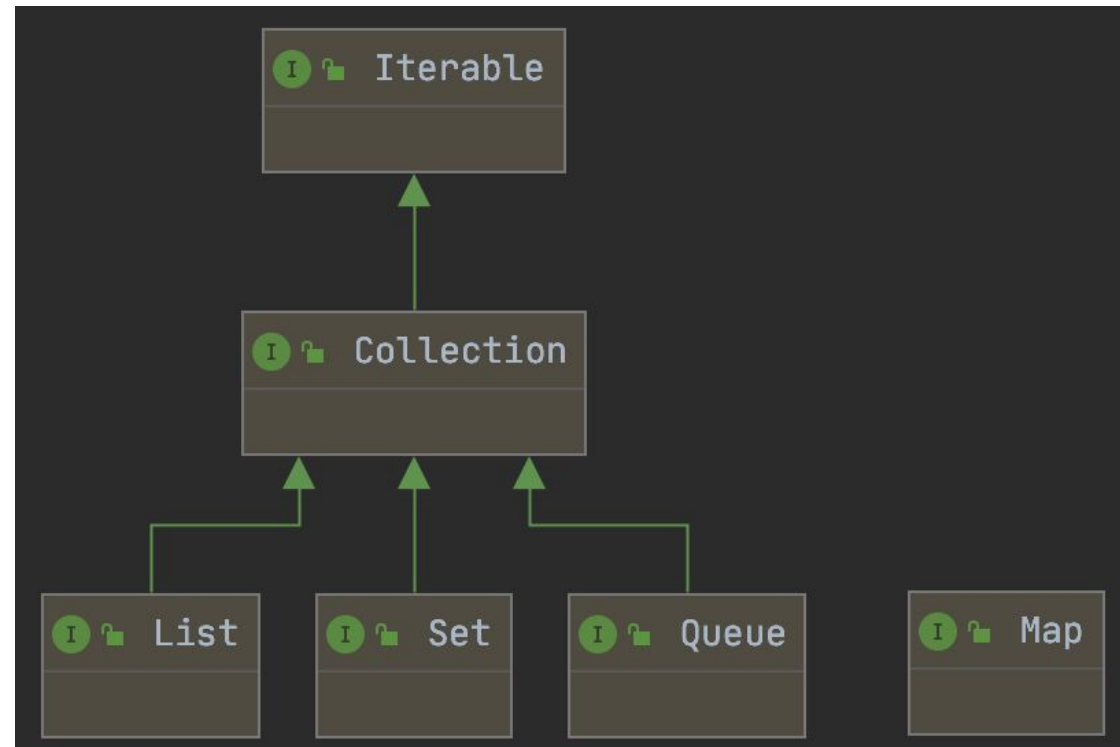
stream(): Stream<E>

parallelStream(): Stream<E>



# Collection Types

- `List<T>` - упорядоченные элементы
- `Queue<T>` - очередь (FIFO)
- `Set<T>` - уникальное множество
- `Map<K, V>` - отображение Key -> Value















# List

```
elif operation == "MIRROR_X":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_o
    #mirror_ob.select = 0
    #time = bpy.context.selected
    #bpy.data.objects[mirror_ob.name].time
```

## List

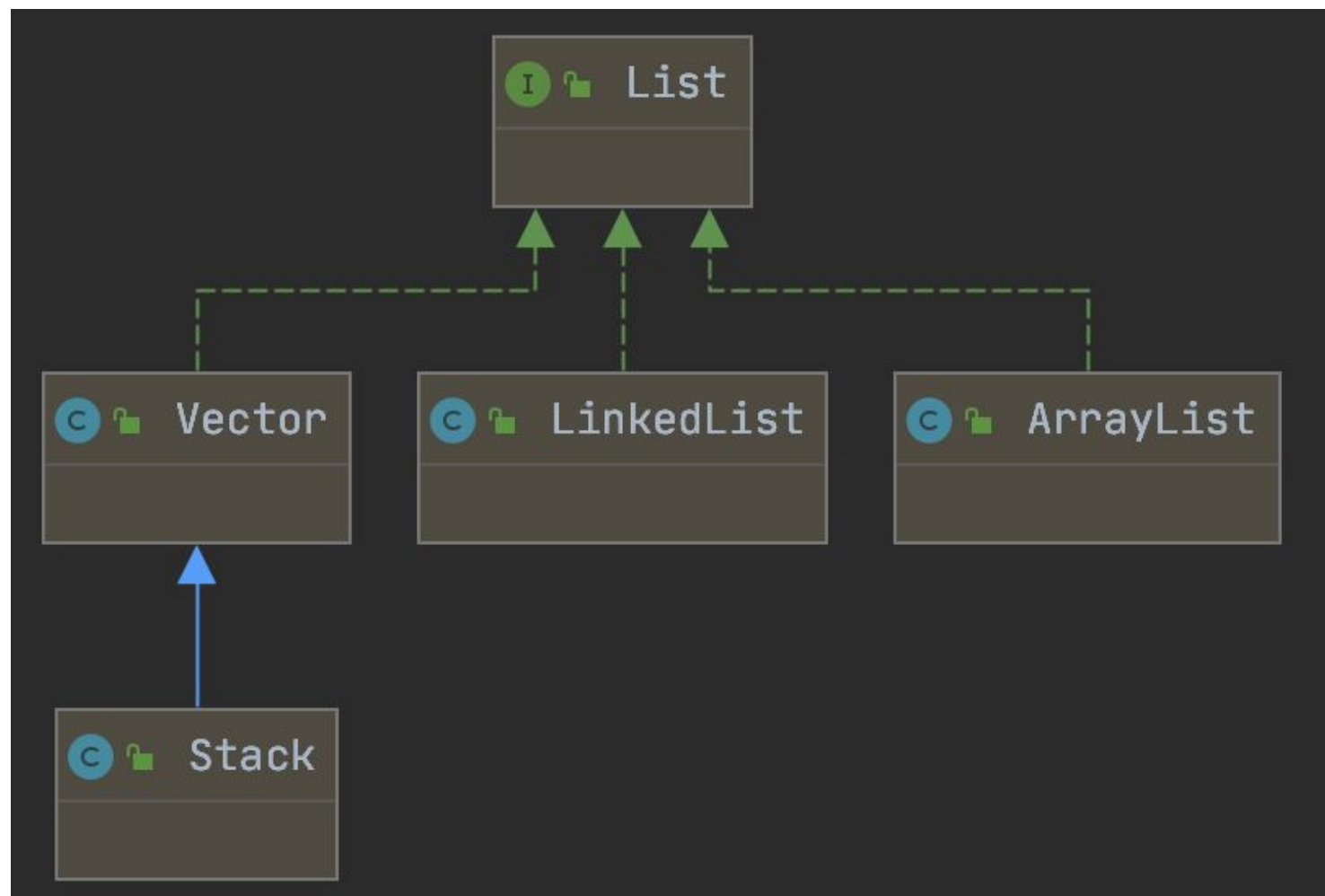
### Collection

- (m)  `addAll(int, Collection<? extends E>): boolean`
- (m)  `replaceAll(UnaryOperator<E>): void`
- (m)  `sort(Comparator<? super E>): void`
- (m)  `get(int): E`
- (m)  `set(int, E): E`
- (m)  `add(int, E): void`
- (m)  `remove(int): E`
- (m)  `indexOf(Object): int`
- (m)  `lastIndexOf(Object): int`
- (m)  `listIterator(): ListIterator<E>`
- (m)  `listIterator(int): ListIterator<E>`
- (m)  `subList(int, int): List<E>`

# List

```
elif _operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif _operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```



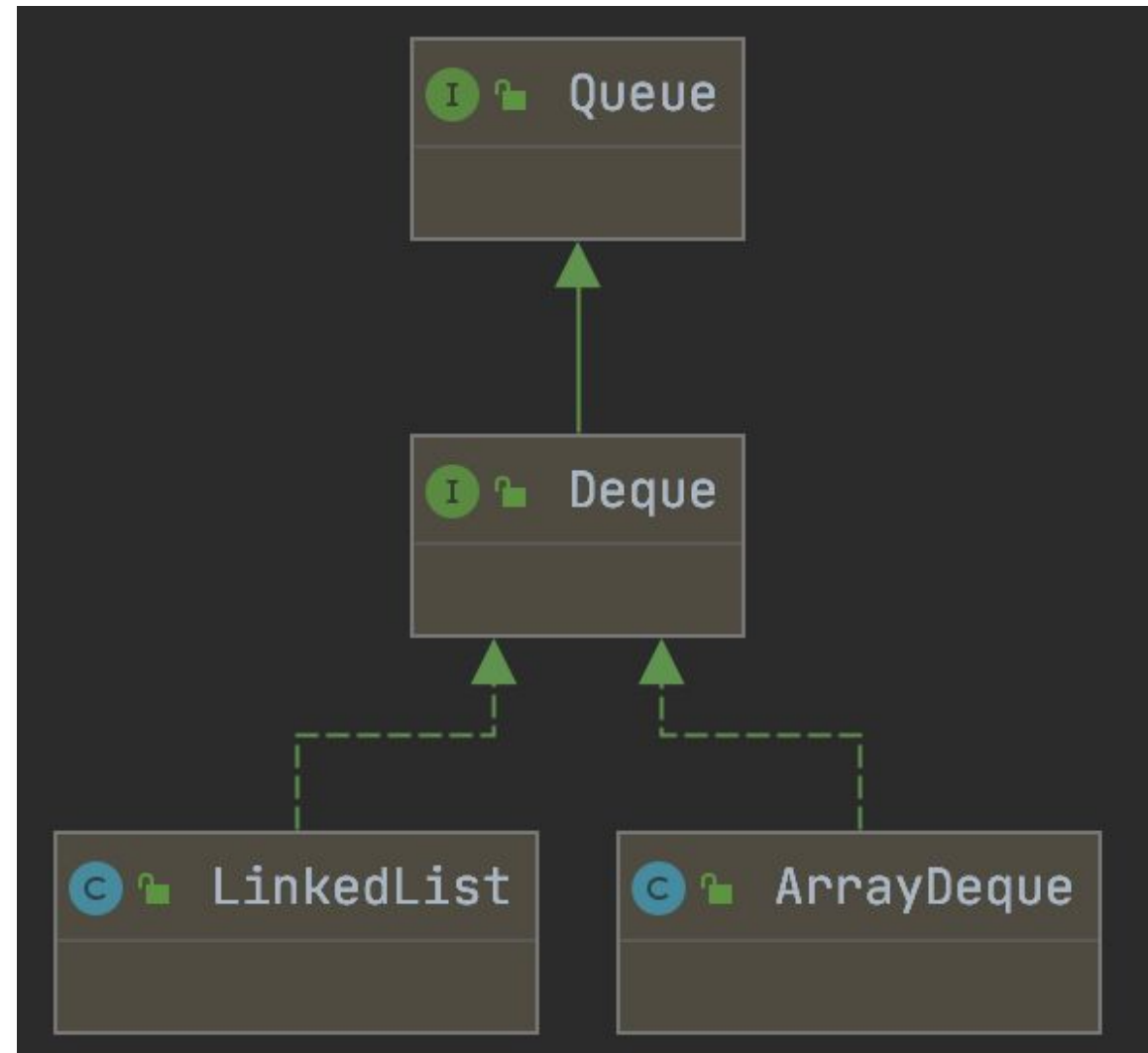


# Queue & Deque

```
Queue
└─ Collection
    └─ offer(E): boolean
    └─ remove(): E
    └─ poll(): E
    └─ element(): E
    └─ peek(): E
```

```
Deque
└─ Queue
└─ Collection
    └─ addFirst(E): void
    └─ addLast(E): void
    └─ offerFirst(E): boolean
    └─ offerLast(E): boolean
    └─ removeFirst(): E
    └─ removeLast(): E
    └─ pollFirst(): E
    └─ pollLast(): E
    └─ getFirst(): E
    └─ getLast(): E
    └─ peekFirst(): E
    └─ peekLast(): E
    └─ removeFirstOccurrence(Object): boolean
    └─ removeLastOccurrence(Object): boolean
    └─ push(E): void
    └─ pop(): E
    └─ descendingIterator(): Iterator<E>
```

# Queue & Deque



# Set

```
elif operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```

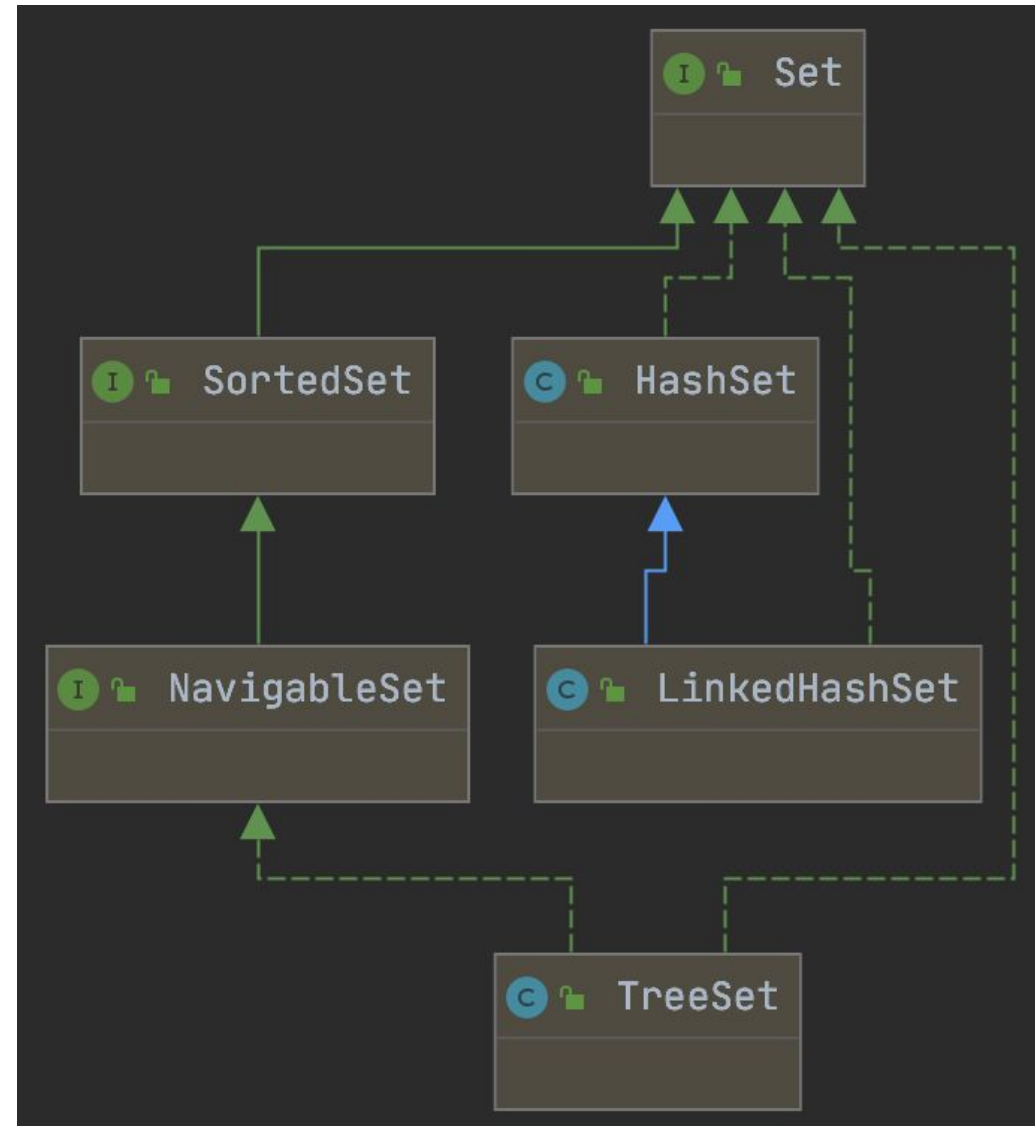
```
Set  
Collection  
of(): Set<E>  
of(E): Set<E>  
of(E, E): Set<E>  
of(E, E, E): Set<E>  
of(E, E, E, E): Set<E>  
of(E, E, E, E, E): Set<E>  
of(E, E, E, E, E, E): Set<E>  
of(E, E, E, E, E, E, E): Set<E>  
of(E, E, E, E, E, E, E, E): Set<E>  
of(E, E, E, E, E, E, E, E, ...): Set<E>  
of(E, E, E, E, E, E, E, E, ...): Set<E>  
of(E, E, E, E, E, E, E, E, ...): Set<E>  
of(E...): Set<E>  
copyOf(Collection<? extends E>): Set<E>
```



# Set

```
elif_operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif_operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```



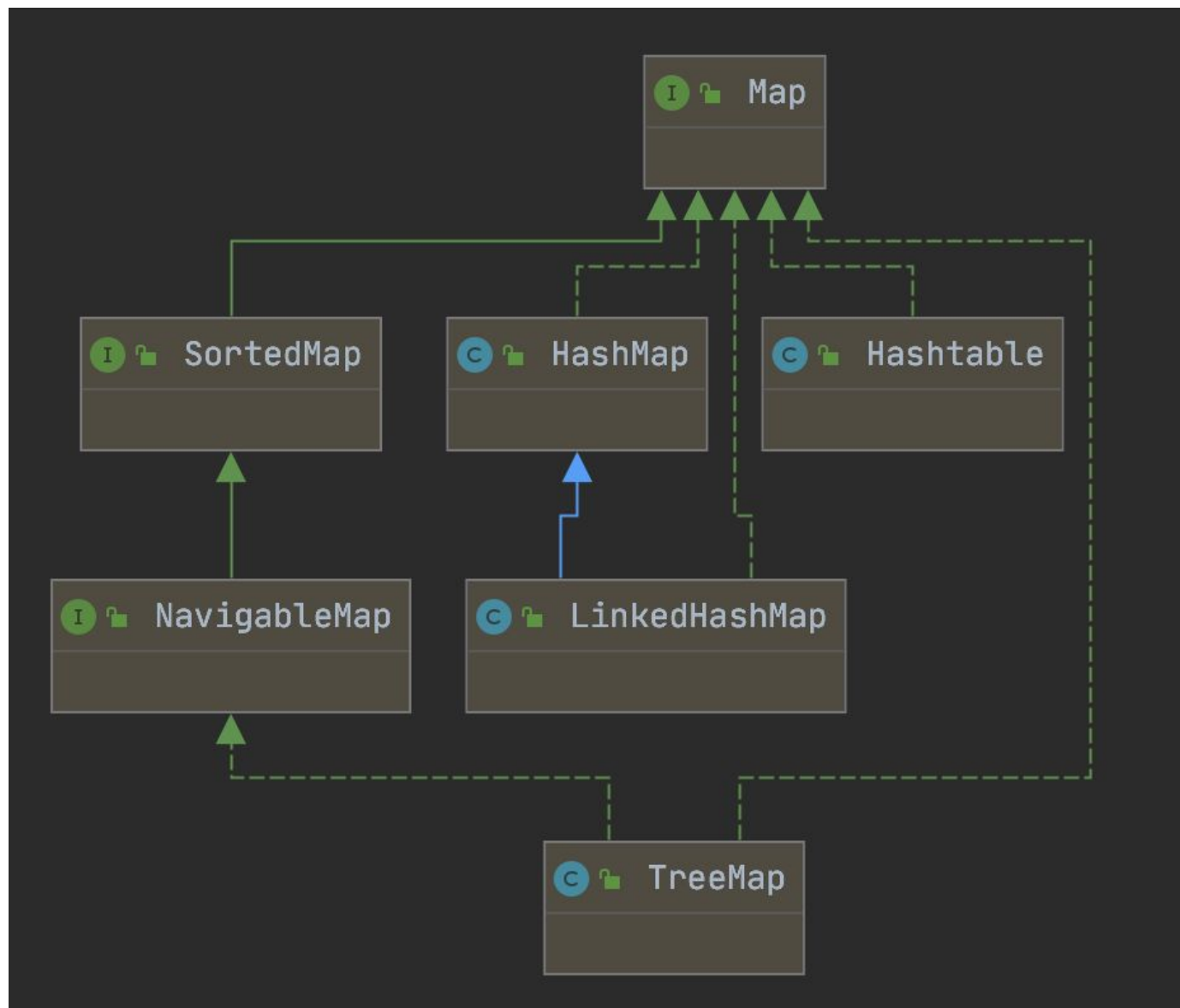
# Map

```
Map
├── Entry
├── Object
│   ├── size(): int
│   ├── isEmpty(): boolean
│   ├── containsKey(Object): boolean
│   ├── containsValue(Object): boolean
│   ├── get(Object): V
│   ├── put(K, V): V
│   ├── remove(Object): V
│   ├── putAll(Map<? extends K, ? extends V>): void
│   ├── clear(): void
│   ├── keySet(): Set<K>
│   ├── values(): Collection<V>
│   ├── entrySet(): Set<Entry<K, V>>
│   ├── getOrDefault(Object, V): V
│   ├── forEach(BiConsumer<? super K, ? super V>): void
│   ├── replaceAll(BiFunction<? super K, ? super V, ? extends V>): void
│   ├── putIfAbsent(K, V): V
│   ├── remove(Object, Object): boolean
│   ├── replace(K, V, V): boolean
│   ├── replace(K, V): V
│   ├── computeIfAbsent(K, Function<? super K, ? extends V>): V
│   ├── computeIfPresent(K, BiFunction<? super K, ? super V, ? extends V>): V
│   ├── compute(K, BiFunction<? super K, ? super V, ? extends V>): V
│   └── merge(K, V, BiFunction<? super V, ? super V, ? extends V>): V
```

# Map

```
elif_operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif_operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True
```

```
#selection at the end -add ba  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active  
print("Selected" + str(modifier_o
```





```

elif operation == "MIRROR Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_o

#mirror_ob.select = 0
name = bpy.context.selected_objects[0].name
bpy.data.objects[name].modifiers[0].name = "Mirror"

```

```

elif operation == "MIRROR Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif operation == "MIRROR Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add ba
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active
print("Selected" + str(modifier_o

#mirror_ob.select = 0
name = bpy.context.selected_objects[0].name
bpy.data.objects[mirror_ob.name].name = name + "Mirror"
print("Mirror added")

```

## Задача 1

Посчитать количество вхождений каждого слова в текст.



## Задача 2

На вход последовательно поступает неограниченное число строк. После каждой 100-ой строки, приходит запрос на получение строк в алфавитном порядке. Необходимо реализовать описанную возможность и, по возможности, минимизировать затрачиваемые ресурсы.



# КРОК

ИНТЕГРИРУЕМ  
БУДУЩЕЕ