# GUAVA

Piotr Kafel

# Guava

- **Na początku były Google Collections...**
  - Rozszerzenie Java Collections Framework
  - Wymaga JDK 1.5+
  - Wersja final - 30 grudnia 2009
  - Zamrożenie API
  - Co dalej ?

# Guava

- No to może Guava...
  - Guava zawiera Google Collections
  - Rozwija idee zawarce w Google Collections
  - Zawiera podstawowe biblioteki wykorzystywane w Google przy projektach Javowych
  - Doskonale udokumentowana
  - Napisana zgodnie z najlepszymi praktykami programistycznymi
  - Aktywnie rozwijany projekt

# Guava

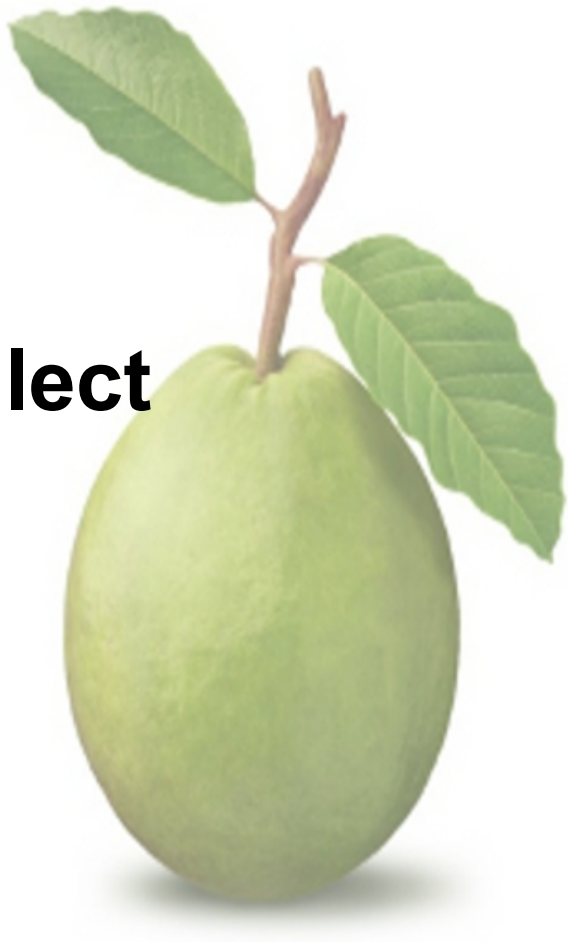
- com.google.common.annotations
- com.google.common.base
- com.google.common.base.internal
- com.google.common.collect
- com.google.common.io
- com.google.common.net
- com.google.common.primitives
- com.google.common.util.concurrent

# Guava

- com.google.common.annotations
- com.google.common.base
- com.google.common.base.internal
- com.google.common.collect
- com.google.common.io
- com.google.common.net
- com.google.common.primitives
- com.google.common.util.concurrent

# Guava

**com.google.common.collect**



# Unmodifiable

## Unmodifiable:

- `Collections.unmodifiableCollection(collection);`
- `Collections.unmodifiableList(list);`
- `Collections.unmodifiableSet(set);`
- …

```
List<Long> myList = new ArrayList<Long>();
myList.add(14L);
myList.add(19L);
List<Long>unmodifiableList = Collections.unmodifiableList(myList);
```

# Immutable

**Unmodifiable ≠ Immutable**

# Immutable

Czy potrzebuję niemutowalnych struktur danych ?

- Ułatwiają programowanie
- Zajmują mniej pamięci
- Są szybsze
- Są łatwe w użyciu
- W większości przypadków kolekcje powinny być niemutowalne
- Bardziej przejrzyste API

# Immutable

```
List<Long> myList = new ArrayList<Long>();
myList.add(14L);
myList.add(19L);
List<Long>unmodifiableList = Collections.unmodifiableList(myList);
```

# Immutable

```
List<Long> myList = new ArrayList<Long>();
myList.add(14L);
myList.add(19L);
List<Long>unmodifiableList = Collections.unmodifiableList(myList);
```

To samo:

```
List<Long> myList = ImmutableList.of(14L, 19L);
```

# Immutable

```
Map<Long, String> myMap = new HashMap<Long, String>();
myMap.put(3L, "What a boring presentation !");
myMap.put(5L, "Yeah... And the speaker is a badass...");
Map<Long,String> unmodifiableMap = Collections
                                        .unmodifiableMap(myMap);
```

# Immutable

```
Map<Long, String> myMap = new HashMap<Long, String>();
myMap.put(3L, "What a boring presentation !");
myMap.put(5L, "Yeah... And the speaker is a badass...");
Map<Long,String> unmodifiableMap = Collections
                                        .unmodifiableMap(myMap);
```

To samo :

```
Map<Long, String> myMap = ImmutableMap.of(3L, "What a boring
presentation !", 5L, "Yeah... And the speaker is a badass...");
```

# Immutable

- `ImmutableList.copyOf();`
- `ImmutableList.of();`

- `ImmutableSet.copyOf();`
- `ImmutableSet.of();`

- `ImmutableMap.copyOf();`
- `ImmutableMap.of();`

# Immutable

Jeżeli nie potrafimy stwierdzić *a priori* jakie elementy znajdą się w kolekcji...

# Immutable

Jeżeli nie potrafimy stwierdzić *a priori* jakie elementy znajdą się w kolekcji...

**Builder !**

```
Builder<Long> builder = ImmutableList.builder();

for(Long someLong : args){
    // do some magic stuff...
    if(...){
        builder.add(someLong);
    }
}

List<Long> myList = builder.build();
```

# Immutable

```
Builder<Long, String> builder = ImmutableMap.builder();

for(Long someLong : args){
    // do some magic stuff...
    if(...){
        builder.put(someLong, someLong.toString());
    }
}

Map<Long, String> map = builder.build();
```

# Immutable

- ImmutableSortedSet
  - `copyOf(Collection )`
  - `copyOf(E[])`
  - `...`
  - `of(E...)`
  - `copyOfSorted(SortedSet)`
  - `...`

  - `builder()`
  - `naturalOrder()`
  - `reverseOrder()`
  - `orderedBy(Comparator)`

# Immutable

- ImmutableSortedMap
  - `copyOf(Map)`
  - `of(K, V)`
  - `copyOfSorted(SortedMap)`
  - `...`

  - `builder()`
  - `naturalOrder()`
  - `reverseOrder()`
  - `orderedBy(Comparator)`

# Immutable

Niemutowalne struktury danych :

- ImmutableList
- ImmutableSet
- ImmutableMap
- ImmutableSortedSet
- ImmutableSortedMap

- ...

# Multimap

Wygląda znajomo ?

```
Map<Long, List<String>> someMap = new HashMap<Long, List<String>>();
```

# Multimap

Wygląda znajomo ?

```
Map<Long, List<String>> someMap = new HashMap<Long, List<String>>();
```

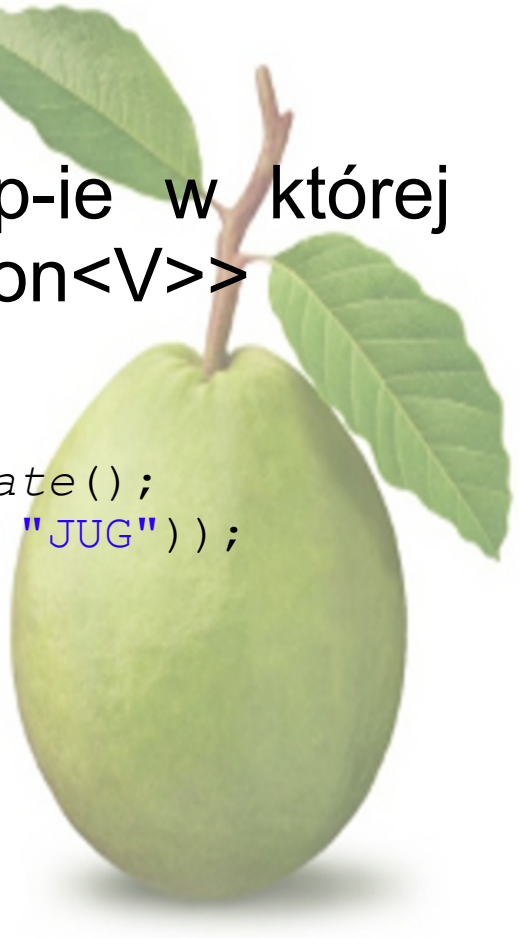
A można tak:

```
Multimap<Long, String> someMap = HashMultimap.create();
```

# Multimap

Multimap - mapa w której jednemu kluczowi można przyporządkować wiele wartości.

O multimap-ie można myśleć jak o map-ie w której wartościami jest kolekcja - Map<K, Collection<V>>



```
Multimap<Long, String> multimap = HashMultimap.create();
multimap.putAll(32L, Lists.newArrayList("Wroclaw", "JUG"));
Collection<String> values = multimap.get(32L);
multimap.remove(32L, "JUG");
multimap.put(32L, "Meeting place");
multimap.removeAll(32L);
```

# Multimap

W większości interfejs multimap-y zgadza się z interfejsem mapy:

```
size(), values(), put(), containsValue(), containsKey()...
```

Niektóre metody uległy zmianie:

- get(K) zwraca Collection<V>
- remove(K, V)
- ...

Doszło również kilka nowych metod:

- removeAll(K)
- keys()
- ...

# Multimap

Dostępne implementacje:

- ArrayListMultimap
- HashMultimap
- ImmutableListMultimap
- ImmutableSetMultimap
- LinkedHashMultimap
- LinkedListMultimap
- TreeMultimap
- ...

# Multiset

## A to wygląda znajomo ?

```
Map<String, Integer> tags = new HashMap<String, Integer>();

public void addTag(String tag){
    if(tags.containsKey(tag)){
        int count = tags.get(tag);
        tags.put(tag, count + 1);
    } else {
        tags.put(tag, 1);
    }
}
```

# Multiset

## A to wygląda znajomo ?

```
Map<String, Integer> tags = new HashMap<String, Integer>();

public void addTag(String tag){
    if(tags.containsKey(tag)){
        int count = tags.get(tag);
        tags.put(tag, count + 1);
    } else {
        tags.put(tag, 1);
    }
}
```

## A można tak:

```
Multiset<String> tags = HashMultiset.create();

tags.add();
```

# Multiset

Multiset – zbiór w którym jeden element może występować wiele razy.

```
Multiset<Long> multiset = HashMultiset.create();
multiset.add(32L);
multiset.add(32L, 5);
multiset.count(32L);
Set<Long> uniqueElements = multiset.elementSet();
multiset.remove(32, 2);
multiset.setCount(32L, 11);
```

# Multiset

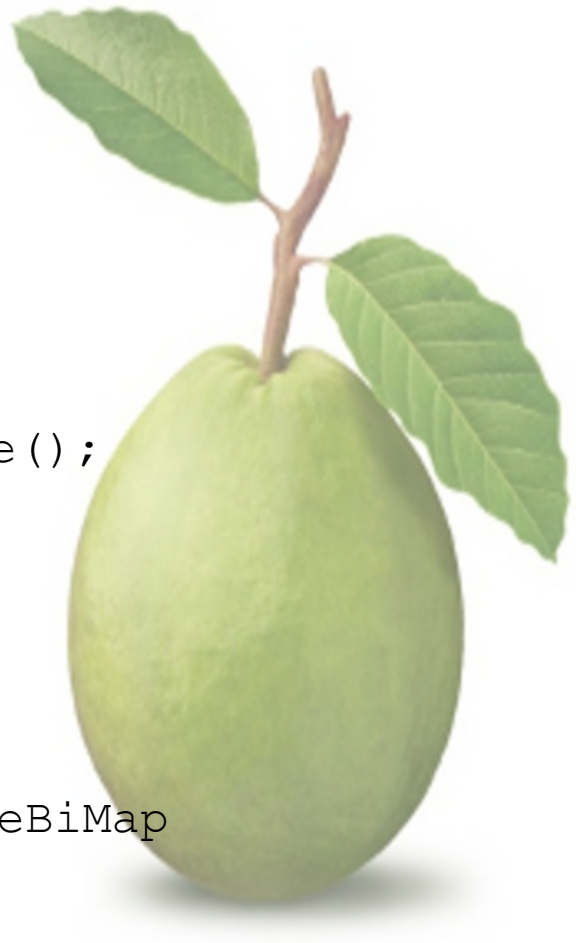
Dostępne implementacje:

- HashMultiset
- ImmutableMultiset
- LinkedHashMultiset
- TreeMultiset
- ...

# BiMap

BiMap – mapa zapewniająca nie tylko unikatowość kluczy, ale również wartości.

```
BiMap<Long, String> biMap = HashBiMap.create();

biMap.put(2L, "Some string");
// biMap.put(3L, "Some string");
biMap.put(23L, "Wroclaw JUG");
biMap.forcePut(11L, "Some string");

BiMap<String, Long> inversedBiMap = biMap.inverse();
```



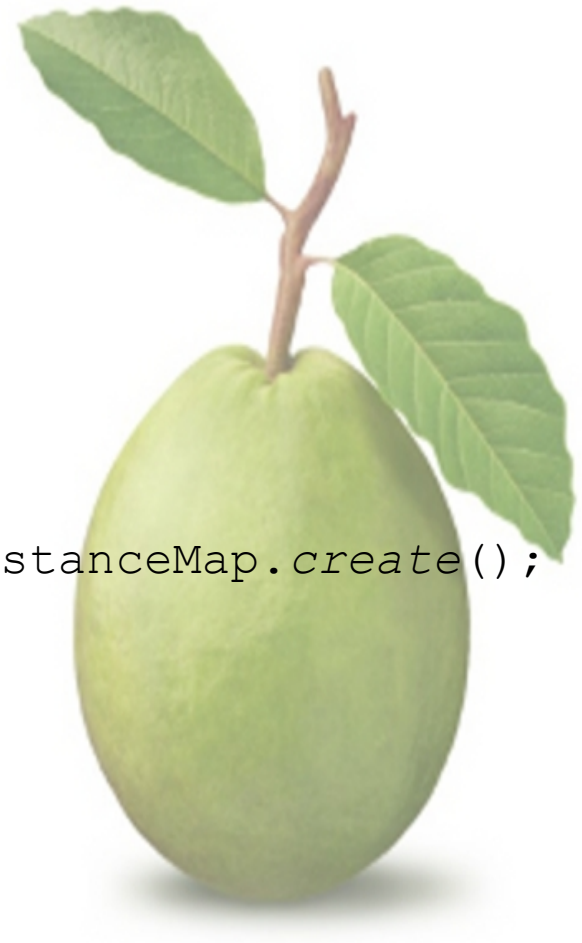
## Dostępne implementacje:

```
EnumBiMap, EnumHashBiMap, HashBiMap, ImmutableBiMap
```

# ClassToInstanceMap

ClassToInstanceMap – mapa przyporządkowująca klasie jej instancje.

```
interface Config{ … }

class ConfigA implements Config{ … }

class ConfigB implements Config{ … }
```

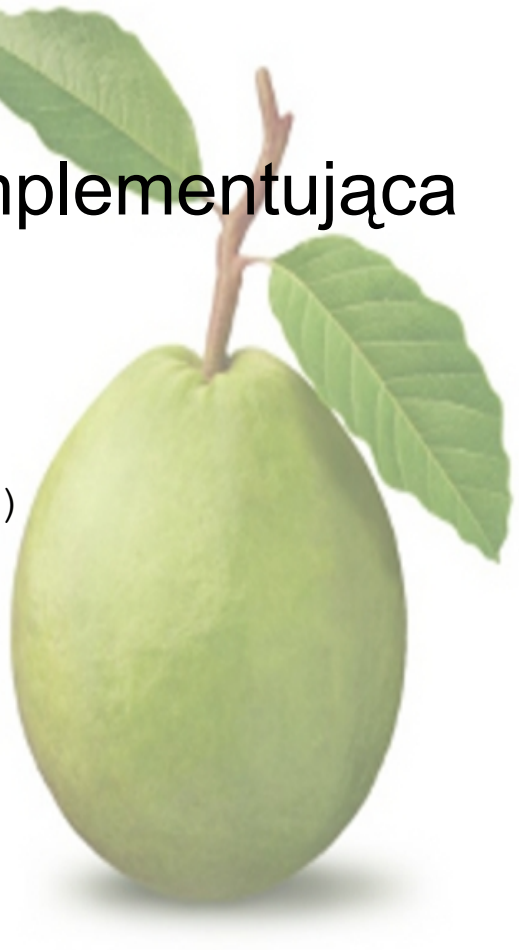


```
ClassToInstanceMap<Config> map = MutableClassToInstanceMap.create();

map.put(A.class, new B());
B a1 = (B)map.get(A.class);

map.putInstance(A.class, new A());
A a2 = map.getInstance(A.class);
```

# ForwardingObject

ForwardingObject – klasa abstrakcyjna implementująca wzorzec projektowy dekorator.

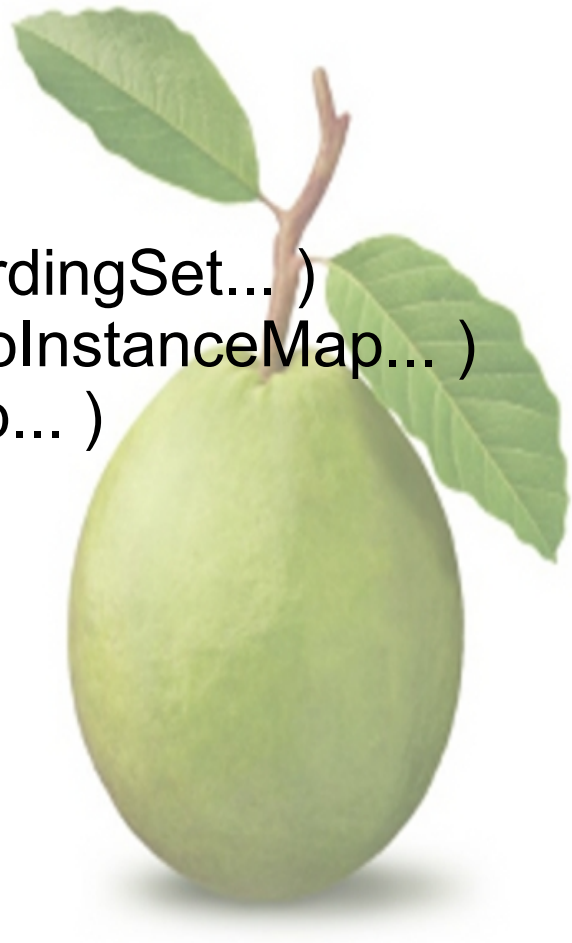
```
protected abstract Object delegate()
```



# ForwardingObject

Klasy dziedziczące po ForwardingObject:

- ForwardingCollection ( ForwardingList, ForwardingSet... )
- ForwardingMap ( HashBiMap, MutableClassToInstanceMap... )
- ForwardingMultimap ( ForwardingListMultimap... )
- ...



# ForwardingObject

```
class MyList<T> extends ForwardingList<T>{

    private List<T> delegate;

    @Override
    protected List<T> delegate() {
        if(delegate == null){
            delegate = Lists.newArrayList();
        }
        return delegate;
    }

    @Override
    public int size(){
        return delegate().size() - 100;
    }
}
```

# Lists, Sets, Maps

- Factory method
  - `Lists.newArrayList()`
  - `Lists.newLinkedList()`
  - `Sets.newHashSet()`
  - `Sets.newLinkedHashSet()`
  - `Sets.newTreeSet()`
  - `Maps.newHashMap()`
  - `Maps.newLinkedHashMap()`
  - …

```
Map<Class<? extends Config>, String> myMap = new HashMap<
                                    Class<? extends Config>, String>();
Map<Class<? extends Config>, String> myMap = Maps.newHashMap();
```

# Function

Java nie posiada funkcji wyższego rzędu (high-order functions) !

```
public interface Function<F, T> {

    T apply(@Nullable F input);

}
```

# Iterators

## Odrobina funkcyjności w Javie !

- `Iterator concat(Iterator...)`
- `boolean contains(Iterator, Object);`
- `Iterator cycle(Iterable);`
- `Iterator filter(Iterator, Predicate)`
- `int frequency(Iterator, Object)`
- `T get(Iterator, int)`
- `T getLast(Iterator)`
- `T getOnlyElement(Iterator)`
- `int size(Iterator)`
- `Iterator transform(Iterator, Function)`

# Iterators, Iterables

Odrobina funkcyjności w Javie !

- `Iterator concat(Iterator...)`
- `boolean contains(Iterator, Object);`
- `Iterator cycle(Iterable);`
- `Iterator filter(Iterator, Predicate)`
- `int frequency(Iterator, Object)`
- `T get(Iterator, int)`
- `T getLast(Iterator)`
- `T getOnlyElement(Iterator)`
- `int size(Iterator)`
- `Iterator transform(Iterator, Function)`

Te same funkcje tylko na Iterable w klasie Iterables !

# Guava

**com.google.common.base**

# Charsets

- Charsets.ISO_8859_1
- Charsets.US_ASCII
- Charsets.UTF_16
- Charsets.UTF_16BE
- Charsets.UTF_16LE
- Charsets.UTF_8

# Joiner

Mechanizm pozwalający na łączenie string-ów.

```
String result = Joiner.on("|").join(stringsToJoin);

String result = Joiner.on("|").skipNulls().join(stringsToJoin);

String result = Joiner.on("|").useForNull("JUG").join(stringsToJoin);
```

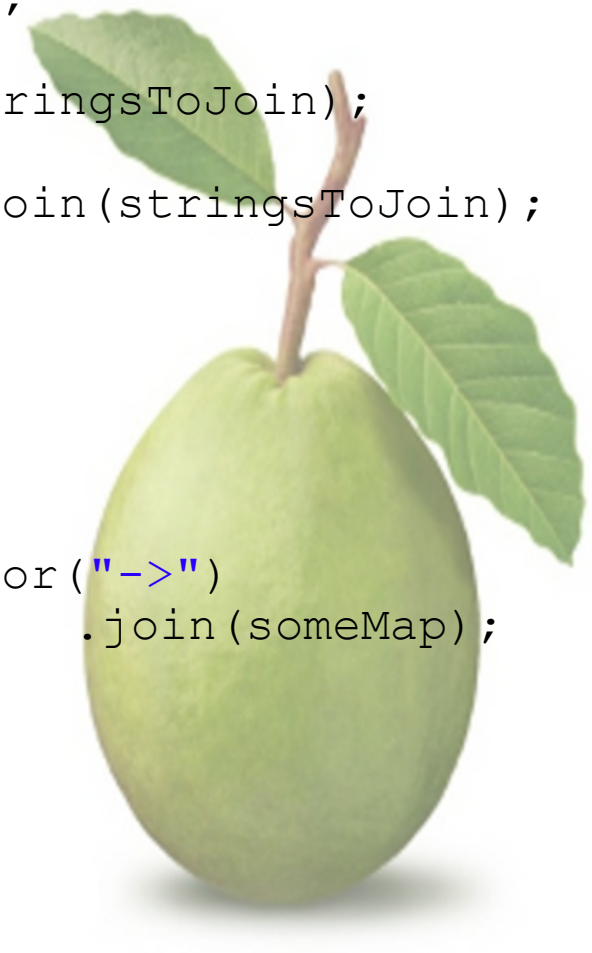
# Joiner

Mechanizm pozwalający na łączenie string-ów.

```
String result = Joiner.on("|").join(stringsToJoin);

String result = Joiner.on("|").skipNulls().join(stringsToJoin);

String result = Joiner.on("|").useForNull("JUG").join(stringsToJoin);
```

Działa też na mapach:

```
String result = Joiner.on("|").withKeyValueSeparator("->")
                              .join(someMap);
```



# Splitter

Mechanizm pozwalający na dzielenie string-ów.

```
Iterable<String> afterSplit = Splitter.on(' ').split("Wroclaw JUG");
```

# Splitter

Mechanizm pozwalający na dzielenie string-ów.

```
Iterable<String> afterSplit = Splitter.on(' ').split("Wroclaw JUG");
```

String w Javie ma metodę split() !

# Splitter

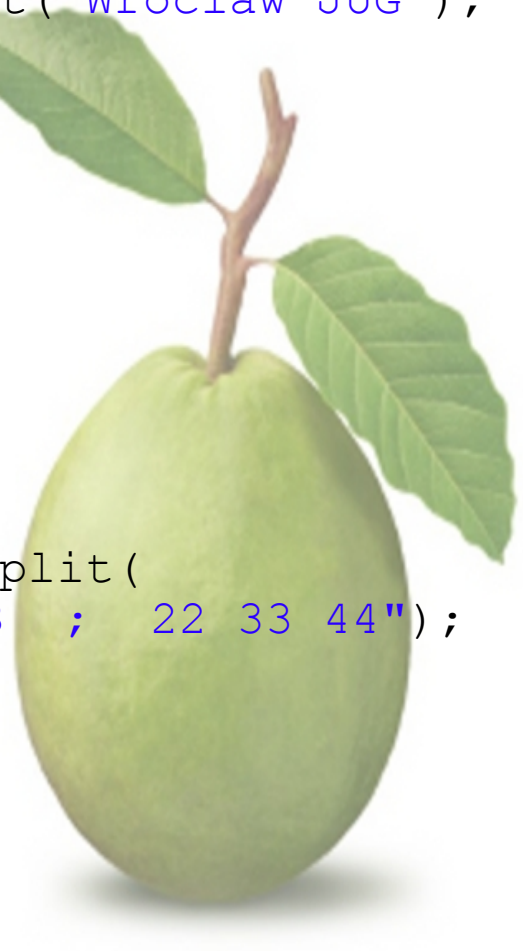
Mechanizm pozwalający na dzielenie string-ów.

```
Iterable<String> afterSplit = Splitter.on(' ').split("Wroclaw JUG");
```

String w Javie ma metodę split() !

Splitter jest dużo bardziej elastyczny.

```
Splitter.on(";").omitEmptyStrings().trimResults().split(
                                "432 432 555; 542233543  ;  22 33 44");
```



# Preconditions

Guava wspiera programowanie defensywne.

```
public void setName(String name){
    if(name == null){
        throw new NullPointerException();
    }
    this.name = name;
}
```

A można tak:

```
public void setName(String name){
    this.name = Preconditions.checkNotNull(name);
}
```

# Preconditions

```
public static void someMethod(Product product) {
    Preconditions.checkNotNull(product);
    Preconditions.checkState(!product.isReadyToSend());
    Preconditions.checkArgument(product.getPrice() > 100.0,
       "Product %s price is %s $. Its is lower then 100$",product
                    .getId(), product.getPrice());

       // do magic stuff
    }
```

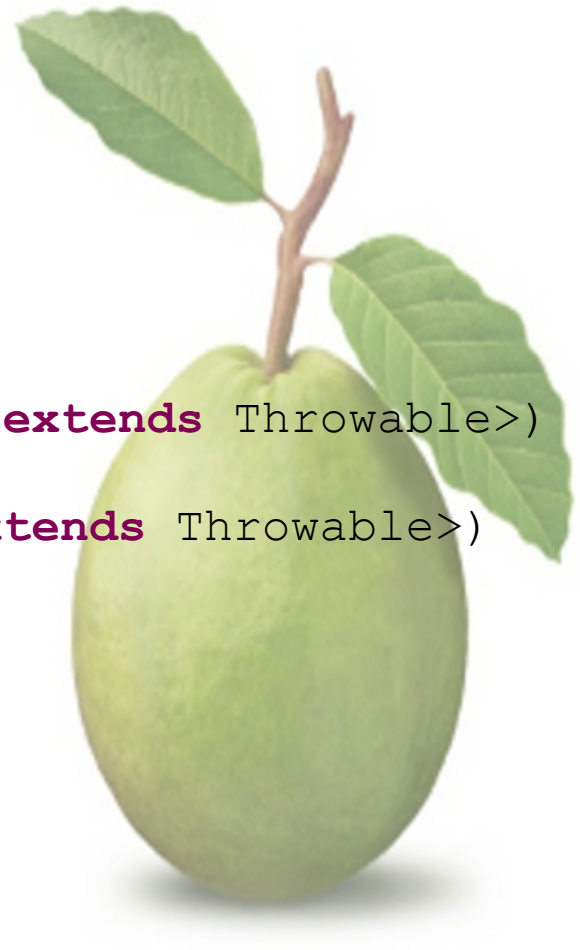
Preconditions.checkNotNull() → NullPointerException
Preconditions.checkState() → IllegalStateException
Preconditions.checkArgument() → IllegalArgumentException

# Throwables

Obsługa klas implementujących Throwable.

- `String getStackTraceAsString(Throwable)`
- `List<Throwable> getCausalChain(Throwable)`
- `Throwable getRootCause(Throwable)`

- `void propagate(Throwable)`
- `void propagateIfInstanceOf(Throwable, Class<X extends Throwable>)`
- `void propagateIfPossible(Throwable)`
- `void propagateIfPossible(Throwable, Class<X extends Throwable>)`



# Guava

**com.google.common.io**

# Files

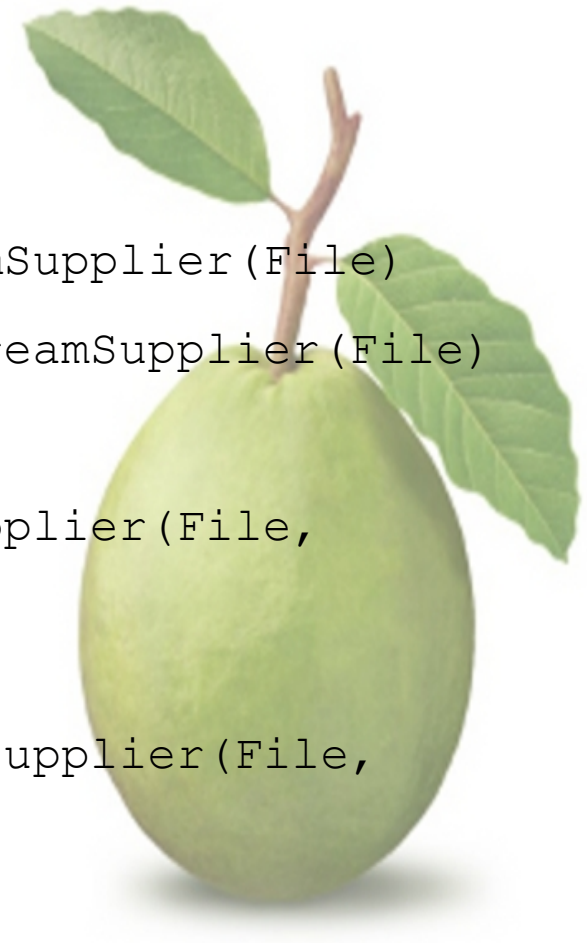
Klasa umożliwiająca operowanie na plikach.

- `void copy(File, File)`
- `void append(CharSequence, File, Charset)`
- `void createParentDirs(File)`
- `void deleteRecursively(File)`
- `void move(File, File)`
- `List<String> readLines(File, Charset);`
- `byte[] toByteArray(File)`
- `String toString(File, Charset)`
- `void write(CharSequence, File, Charset);`
- `...`

# Files

```
public interface InputSupplier<T> {

    T getInput() throws IOException;

}
```

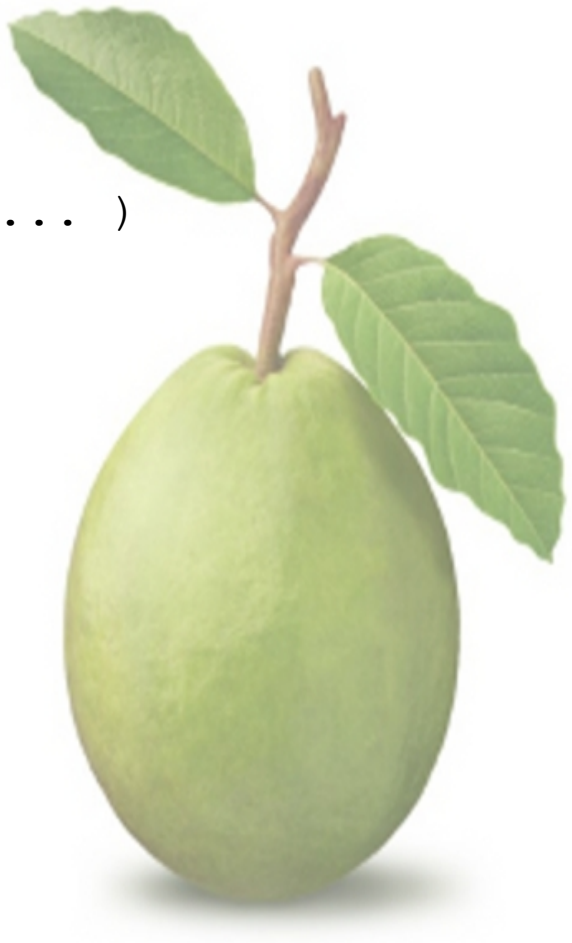
- InputSupplier<FileInputStream> newInputStreamSupplier(File)
- OutputSupplier<FileOutputStream> newOutputStreamSupplier(File)
- BufferedReader newReader(File, Charset)
- InputSupplier<InputStreamReader> newReaderSupplier(File, Charset)
- BufferedWriter newWriter(File, Charset)
- OutputSupplier<OutputStreamWriter> newWriterSupplier(File, Charset)
- ...



# ByteStreams

Klasa ułatwiająca pracę na strumieniach bajtów.

- `long copy(InputStream, OutputStream)`
- `InputSupplier<InputStream> join(InputSupplier... )`
- `void readFully(InputStream , byte[])`
- `void skipFully(InputStream, long)`
- `byte[] toByteArray(InputStream)`
- `...`



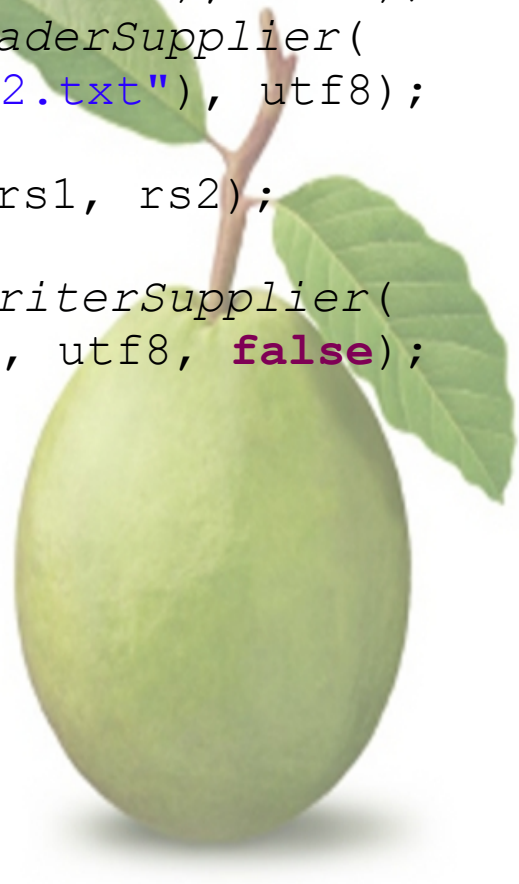
# CharStreams

Klasa ułatwiająca pracę na strumieniach znaków.

- `long copy(InputSupplier, OutputSupplier)`
- `InputSupplier join(InputSupplier...)`
- `void skipFully(Reader, long)`
- `String toString(InputSupplier)`
- `void write(CharSequence, OutputSupplier)`
- `...`

# CharStreams

```
Charset utf8 = Charset.forName("UTF-8");
InputSupplier<InputStreamReader> rs1 = Files.newReaderSupplier(
                                new File("data", "test1.txt"), utf8);
InputSupplier<InputStreamReader> rs2 = Files.newReaderSupplier(
                                new File("data", "test2.txt"), utf8);

InputSupplier<Reader> combined = CharStreams.join(rs1, rs2);

OutputSupplier<OutputStreamWriter> ws = Files.newWriterSupplier(
                                new File("output.txt"), utf8, false);

CharStreams.copy(combined, ws);
```



# ByteArrayDataOutput

```
ByteArrayDataOutput byteOut = ByteStreams.newDataOutput();
byteOut.writeDouble(12.4);
byteOut.writeInt(32);
byte[] bytes = byteOut.toByteArray();
```

# ByteArrayDataOutput

```
ByteArrayDataOutput byteOut = ByteStreams.newDataOutput();
byteOut.writeDouble(12.4);
byteOut.writeInt(32);
byte[] bytes = byteOut.toByteArray();
```

Ta sama funkcjonalność przy odczycie:

**ByteArrayDataInput !**

# CountingOutputStream

```
OutputSupplier<FileOutputStream> supplier = Files
                .newOutputStreamSupplier(new File("output.dat" ));
CountingOutputStream cos = new CountingOutputStream( supplier
                                                    .getOutput() );
String testString = "TEST STRING";
cos.write( testString.getBytes( Charset.defaultCharset() ) );

long byteWrote = cos.getCount();
```

# CountingOutputStream

```
OutputSupplier<FileOutputStream> supplier = Files
                .newOutputStreamSupplier(new File("output.dat" ));
CountingOutputStream cos = new CountingOutputStream( supplier
                                                    .getOutput() );
String testString = "TEST STRING";
cos.write( testString.getBytes( Charset.defaultCharset() ) );

long byteWrote = cos.getCount();
```

Ta sama funkcjonalność przy odczycie:

**CountingInputStream !**

# Guava

I to tyle ?

O czym nie powiedziałem :

*CharMatcher, Ordering, Table, Objects,MapMaker MinMaxPriorityQueue …*

http://code.google.com/p/guava-libraries/

# Guava

Q & A