OOP ŠABLONE

①
```
@Override
public int hashCode() {   hashed1, ..., hashedN
    return Object.hash (...);
}

@Override
public boolean equals (Object obj) {
    if (!(obj instanceOf KLASA))
        return false;
    KLASA other = (KLASA) obj;
    return this.hashed1.equals (other.hashed1) && (...)
          this.hashedN.equals(other.hashedN);

@Override
public int compareTo (KLASA other) {
    int r;
    r = this.hashed1.compareTo (other.hashed1);
    if (r!=0)
        return r;
    (...)
    else
        return this.hashedN.compareTo (other.hashedN);
```

② COMPARATOR
```
public class KLASA implements Comparable <KLASA> {...}

public static Comparator <KLASA> By_NUM = (a,b) -> a.data - b.data;
public static Comparator<    > By_STRING = (a,b) -> comparator.compare (a.string - b.string);
```

③ ITERATOR - LISTA

```
public class KLASA1 <T extends KLASA> implements Iterable <T> {
    ...
    public void add(T t) {
        List <T> aList = aCollection. get (t. getType ());
        if (aList == null) {
            aList = new ArrayList <>();
        }
        if (!aList. contains (t)) {
            aList. add (t);
        }
        aCollection. put (t. getType (), aList);
    }

    public void add (T... elements) {
        for (T t: elements) {
            add(t);
        }
}
private class MyIterator implements Iterator <T> {
        ...         lists ;
        int current = -1 ;

    private MyIterator () {
        lists = new ArrayList <> (aCollection. size());
        for (List <T> list : aCollection. values ()) {
            list. add (list. iterator ());
        }
    }

    @Override
    public boolean hasNext () {
        for (Iterator <T> it : lists) {
            if (it. hasNext ()) {
                return true;
            }
        }
        return false;
    }

    @Override
    public T next () {
        if ( hasNext()) {
            T result;
            while (true) {
                current = ( current + 1 ) % lists. size();
                if (lists. get (current ). hasNext ()) {
                    result = lists. get (current ). next ();
                    break ;
                }
            }
            return result;
        } else {
            throw new NoSuchElementException ();
        }
    }
}
```

4. ITERATOR – MAPE <K, V>

```
                                                       može biti: KLASA, Integer, ...
public class (...) implements Iterable <Pair <K, V>>
    private Map <K, V> map = new TreeMap <> (Comparator.<K> naturalOrder().reversed());

    public void add (K k) {
        V count = map.get (k);
        count = count == null ? 1 : ++count;
        map.put (k, count);
    }

    public void add(K...k) {
        for (K k : ks)
            add (k);
    }

    public void remove (K k) {
        V count = map.get(k);
        if (count != null) {
            --count;
            if (count == 0)
                map.remove(k);
            else
                map.put (k, count);

    @Override
    public Iterator <Pair <K, V>> iterator () {
        return new MyIterator();
    }
}

private class MyIterator implements Iterator <Pair<K, V>> {

    private Iterator <Map.Entry <K,V>> iterator;
    public MyIterator () {
        iterator = map.entrySet().iterator();
    }

    @Override
    public boolean hasNext () {
        return iterator.hasNext();
    }

    @Override
    public Pair <K, V> next() {
        var next = iterator.next();
        return new Pair <> ( next.getKey(), next.getValue());
    }
}
```

⑤ KOLEKCIJSKI TOKOVI — MAPE

```
// u Main, prirutanje mape

return aMap. entrySet()
             . stream()
             . map (aMapEntry -> new SimpleEntry (aMapEntry . getKey(),
                                 aMapEntry . getValue (). values(). stream().
//* optional                  * mapToDATATYPE ( DATATYPE2 :: value ). sum ())) -> sum, avg,..
             . collect (Collectors. toMap (SimpleEntry :: getKey(), SimpleEntry :: getValue());


// pod klasu
// napuni mapu
 aMap . values()
      . stream()
      . flatMap( dataMap -> dataMap . entrySet(). stream())
      . forEach ( dataPair -> tempMap.merge ( dataPair . getKey () , ...),
                  return ... );     // može biti svašta

//temp map , ako ga treba

return tempMap . entrySet()
               . stream ()
               . map ( entry -> new SimpleEntry ( entry. get key(),
                               entry.getValue (). stream . MapToDATA(D :: v)
                                                              DATA
                                         . average (). ~~getAs~~ Double ()))
               . collect ( Collectors . toMap ( SimpleEntry::getKey , SimpleEntry :: getValue));
```

⑥ KOLEKCIJSKI TOKOVI – LISTE

//printaj listu

```
alist . stream ()
      . filter (p)   // p može biti uvjet, npr.  a -> a . getData () >= 1
      . map (a -> a . getData ())    // ako je u zadatku i mapa
      . sorted ()       // u () može biti npr. (a, b) -> a.getData ().compareTo (b.getData())
      . distinct ()
      . forEach ( a -> System . out . println(a));
```

⑦ FILEOVI

// u main, visitor

```
DATATYPE something = visitor. getMetaData ()   // može i npr. getDeletedData()
                              . entrySet()
                              . stream ()
                              . map( s -> s. doSomething ())  -> npr. getValue()
                              . mapTo DATATYPE ( DATATYPE :: typeValue)
                              . sum () / average ()/ something () ;

System.out.format ("          " , something, visitor. get_Data. size() )
                                              ako treba file size
```

7 GRAFIČKA SUČELJA

```
//ako treba napraviti frame u mainu

JFrame frame = new LetterCounterFrame();
frame.setTitle("title");
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
frame.setLocation(100, 100);
frame.setVisible(true);
frame.pack();
// -> ovo dolje je ili u mainu ili u Frame klasi
// za svaki panel koji treba
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());   -> ili GridLayout() ili...
add(panel, BorderLayout.NORTH);  -> ili neka druga strana

// ako ima scroll
JScrollPane scroll = new JScrollPane();
add(scroll, BorderLayout.CENTER);

//dodaj button
JPanel southPanel = new JPanel();
add(southPanel, BorderLayouth.SOUTH);
JButton button = new JButton("Title");

//button listener
button.addActionListener(ActionEvent e) {
        // @Override
        // public void ActionPerformed(ActionEvent e) {
                button.setEnabled(false);          } ako se gumb ne može ponovno
                textArea.setText("   ");           } kliknuti prije kraja eventa
                setText(text) / setValue(0);
                WorkerKlasa worker = new WorkerKlasa(message);
                worker.execute;
        }
}

//ako ima Worker
private class WorkerKlasa extends SwingWorker <Long, String> {

        @Override
        protected void process(List<String> chunks){
                for (String chunk : chunks)
                        textArea.append(chunk + "\n");  //ili progrBar.setValue(get+1);
        }

        @Override
        protected void done(){
                print | setText() | textArea.append;
                button.setEnabled(true);
        }

        @Override
        protected Long doInBackground() throws Exception {
                //ono što se događa kad kliknemo gumb
                return rezultat;
        }
}
```