# Bilješke OOP

SwingWorker example

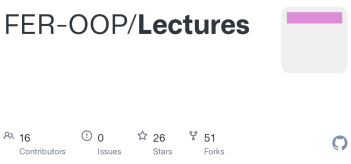
```
class ImageLoader extends SwingWorker<String, String> {
    @Override
    protected String doInBackground() throws Exception {
      List<String> toLoad = new ArrayList<>();
      if(cbEuropean.isSelected()) {
        toLoad.addAll(EUROPEAN);
      }
      if(cbPacific.isSelected()) {
        toLoad.addAll(PACIFIC);
      }
      if(toLoad.isEmpty()) {
        return "Nothing to load.";
      }
      int lengthOfTask = toLoad.size();
      int current = 0;
      setProgress(0);
      for(String country: toLoad) {
        try {
          String imageAddress = String.format("https://cdn.countryflags.com/thumbs/%s/flag-400.png", country.replace(' ', '-').toLower
                    URL url = new URL(imageAddress);
                    BufferedImage image = ImageIO.read(url);
                    images.put(country, new ImageIcon(image));
        } catch (Exception exp) {
          return "Problems with network";
                }
                current++;
                setProgress(100 * current / lengthOfTask);
                publish(country);
                Thread.sleep(50); // to be slower

      }
      return "Finished";
    }
    @Override
    protected void process(List<String> countries) {
      for(String country: countries) txStatus.setText("Downloading " + country);
      model.addAll(countries);
    }
    @Override
    protected void done() {
      try {
        txStatus.setText(get());
      } catch (InterruptedException | ExecutionException e) {
        e.printStackTrace();
      }
      setCursor(null);
      btLoad.setEnabled(true);
      countries.setSelectionInterval(0, 0);
      progressBar.setValue(0);
    }
  }
```

FER-OOP/Lectures

Contribute to FER-OOP/Lectures development by creating an account on GitHub.

https://github.com/FER-OOP/Lectures/blob/master/Exercises/Homework-14/src/main/java/hr/fer/oop/homework_13_e8/FlagsAgain.java




Listener example

```
class MultiListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      int value;
      if(e.getSource() == btRed) {
        value = Integer.valueOf(lbRed.getText()) + 1;
        lbRed.setText(String.valueOf(value));
      }
      if(e.getSource() == btBlue) {
        value = Integer.valueOf(lbBlue.getText()) + 1;
        lbBlue.setText(String.valueOf(value));
      }
    }
  }
```

Compute and function example

```
Integer newGrade = grades.compute("Ante",
  new BiFunction<String, Number, MojInt>() {
  @Override
  public MojInt apply(String t, Number u) {
    return u==null ? 1 : u+1;
  }
});
```

Merge example

```
Integer newGrade = grades.merge("Ante", 1, (oldValue,value) ->
oldValue + value);
```

Lambda, Consumer and Stream example

```
List<Student> students = StudentData.load();
// using anonymous class
students.stream().forEach(new Consumer<Student>() {
@Override
public void accept(Student t) {
System.out.println(t);
}
});
// using lambda
students.stream().forEach(t -> System.out.println(t));
```

Filter example

```
List<Student> students = StudentData.load();
students.stream()
.filter(s -> s.getPoints() >= 5)
.forEach(t -> System.out.println(t));
```

```
Stream<T> filter(Predicate<? super T> predicate);
```

Comparator.comparing definition and example

```
comparing(Function<? super T,? extends U> keyExtractor)
```

```
titleVotes.entrySet().stream().
sorted(Comparator.comparing(Entry<String, Integer>::getValue).reversed()).
```

```
collect(Collectors.toList());
```

SortPerVotes without another map example

```
private static List<Entry<String, Integer>> getTitlesSortedPerVotes(Map<PollingStation,
 Map<String, Integer>> results) {
        return results.
                entrySet().
                stream().
                map(entry -> new SimpleEntry<>(entry.getKey().getTitle(),
                entry.getValue().values().stream().mapToInt(Integer::intValue).sum())).
                collect(Collectors.toMap(Entry::getKey, Entry::getValue, Integer::sum)).
                entrySet().
                stream().
                sorted(Comparator.comparing(Entry<String, Integer>::getValue).reversed()).
                collect(Collectors.toList());
    }
```

Creating custom classes:

- must implement and override equals(x)
- must implement and override hashCode()

Equals and hachCode with hash arguments example

```
@Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Room other = (Room) obj;
        if (Double.doubleToLongBits(this.area) != Double.doubleToLongBits(other.area)) {
            return false;
        }
        if (this.heated != other.heated) {
            return false;
        }
        if (this.numberOfWindows != other.numberOfWindows) {
            return false;
        }
        if (this.floorType != other.floorType) {
            return false;
        }
        if (!Objects.equals(this.wallColor, other.wallColor)) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        return Objects.hash(this.area, this.floorType, this.heated, this.numberOfWindows, this.wallColor);
    }
```

When using Tree—-something collections for custom classes, must implement comparator or comparable interface

```
//Definition Comparable and Comparator

public interface Comparable<T> {
public int compareTo(T o);
}

public interface Comparator<T> {
```

```
int compare(T o1, T o2);

}
```

Example with Comparable

```
public class Student5 extends Student4 implements
Comparable<Student5> {
...
@Override
public int compareTo(Student other) {
return this.studentID.compareTo(other.studentID);
}
}
```

Example with Comparator

```
public class StudentComparator implements Comparator<Student> {
@Override
public int compare(Student s1, Student s2) {
return s1.getStudentID().compareTo(s2.getStudentID());
}
}
```

Reverse order example

```
Comparator<Student> reverse = comparator.reversed();

Comparator<Student> reverse =
Collections.reverseOrder(comparator);

Comparator<Student> reverse = Collections.reverseOrder();
```

Built.in methods to imitate Composite Comparator

```
public static void main(String[] args) {
Comparator<Student8> comparator = Student8.BY_FIRST_NAME.reversed()
.thenComparing(Student8.BY_LAST_NAME)
.thenComparing(Comparator.naturalOrder());
Set<Student> students = new TreeSet<>(comparator);
...
}
```

Iterable and Iterator definition

```
// Iterable<T> implements:
public Iterator<T> iterator()

// Iterator<T> implements:
public boolean hasNext()
public T next()
```

Iterable and Iterator example

Predicate definition

```
// Predicate<T> has to implement:
boolean test(T t);
```

Lambda expression in function of Predicate example

```
printCars(cars, (Car car) -> car.getType() == CarType.DIESEL);
```

File constructors

```
File(String pathname)
  new File("d:/tmp/readme.txt")

File(String parent, String child)
  new File("d:/tmp", "readme.txt")

File(File parent, String child)
  File dir = new File("d:/tmp");
  File file = new File(dir, "readme.txt");

File(URI uri)
  new File(new URI("file:///d:/tmp/readme.txt"));
```

FileVisitor example

```
public class VotingResultVisitor extends SimpleFileVisitor<Path>  {
  private Map<String, Integer> standings = new HashMap<>();
  @Override
  public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
    String filename = file.getFileName().toString();
    if (filename.endsWith("-jury.txt") || filename.endsWith("-televoting.txt")) {
      PointsUtil.updateStandings(file, standings);
    }
    return FileVisitResult.CONTINUE;
  }

  public Map<String, Integer> standings() {
    return standings;
  }
}
```

To read: Input and Output Stream and filters

Good to know

## Koju vrstu klasa kada koristiti?

- Lambda izraz
  - za definiranje „jednostavnog” ponašanja (npr. opis što napraviti sa svakim elementom iz skupa) koje treba proslijediti negdje drugdje u kodu
- Anonimna klasa
  - u slučajevima kad lambda izraz nije prikladan, jer treba definirati dodatne atribute ili metode
- Lokalna klasa
  - kad se klasa ne koristi nigdje van metode u kojoj je definirana, ali postoji više instanci klase, trebamo konstruktor ili je jednostavno potreban imenovani tip zbog dodatnih metoda ili varijabli
- Ugniježđene statičke klase
  - u slučajevima sličnim lokalnoj klasi, ali kad je potrebna šira vidljivost klase
- Unutarnja klasa:
  - kad je potrebno pristupati privatnim članovima vanjske klase