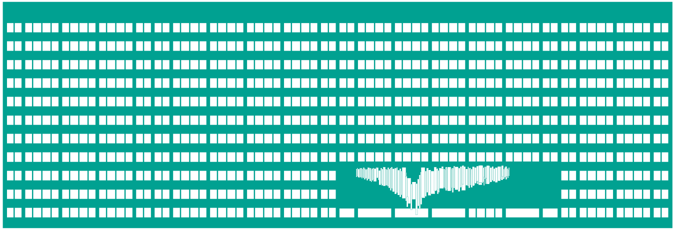
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

# Cviceni 5

Lukas Tomaszek

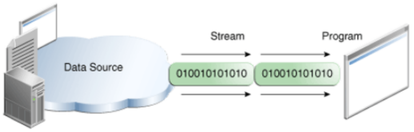
VSB – Technical University of Ostrava

lukas.tomaszek@vsb.cz

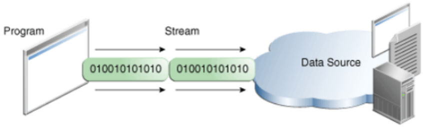
October 31, 2019

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

## Stream



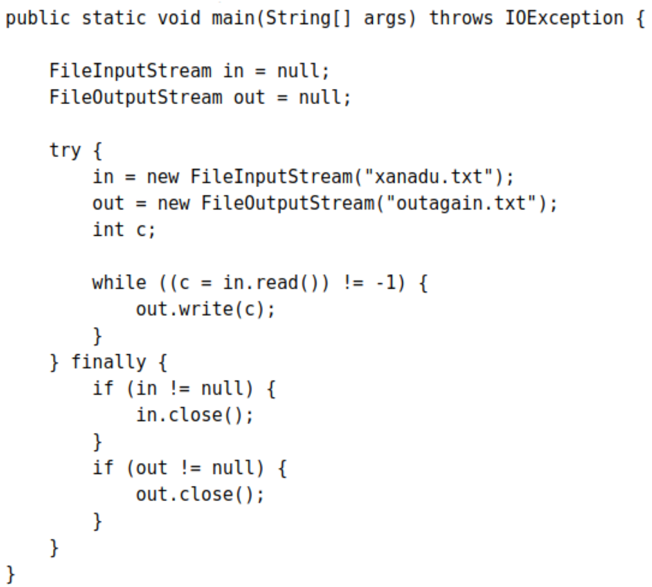
Reading information into a program.



Writing information from a program.

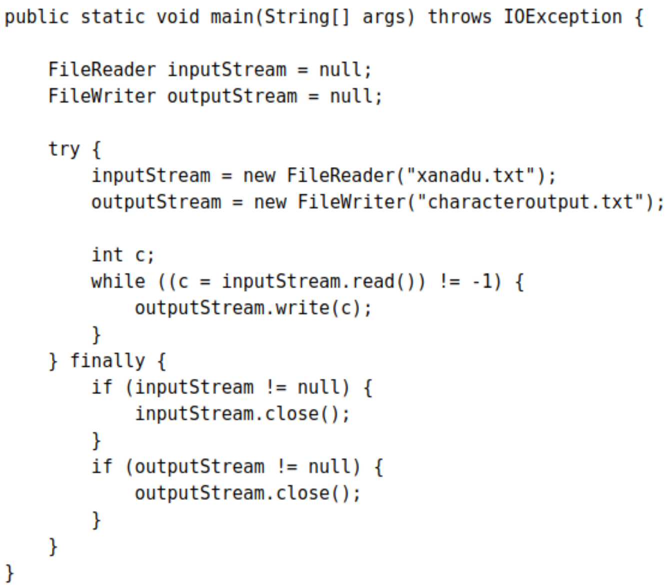
- A stream is a sequence of data.
- https://docs.oracle.com/javase/tutorial/essential/io/index.html

## Byte Stream



```java
public static void main(String[] args) throws IOException {

    FileInputStream in = null;
    FileOutputStream out = null;

    try {
        in = new FileInputStream("xanadu.txt");
        out = new FileOutputStream("outagain.txt");
        int c;

        while ((c = in.read()) != -1) {
            out.write(c);
        }
    } finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
    }
}
```

## Character Stream

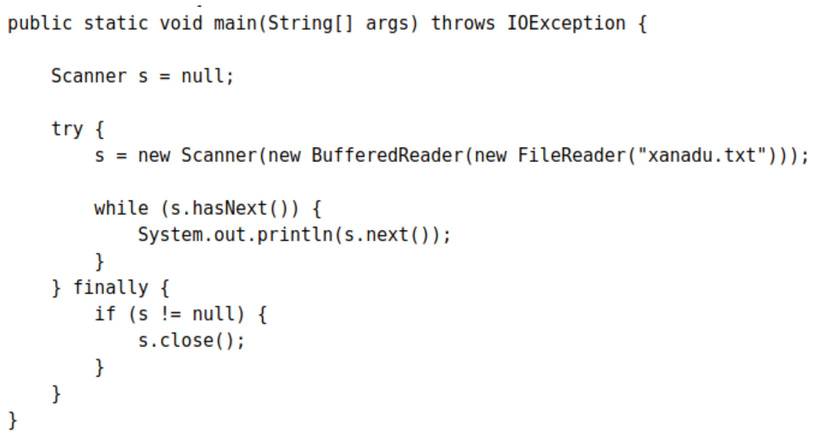


```
public static void main(String[] args) throws IOException {

    FileReader inputStream = null;
    FileWriter outputStream = null;

    try {
        inputStream = new FileReader("xanadu.txt");
        outputStream = new FileWriter("characteroutput.txt");

        int c;
        while ((c = inputStream.read()) != -1) {
            outputStream.write(c);
        }
    } finally {
        if (inputStream != null) {
            inputStream.close();
        }
        if (outputStream != null) {
            outputStream.close();
        }
    }
}
```

## Buffered Stream

```
inputStream = new BufferedReader(new FileReader("xanadu.txt"));
outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));
```

## Scanner

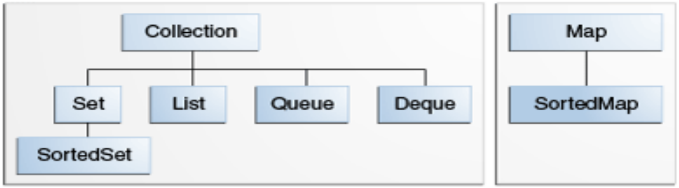


```
public static void main(String[] args) throws IOException {

    Scanner s = null;

    try {
        s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));

        while (s.hasNext()) {
            System.out.println(s.next());
        }
    } finally {
        if (s != null) {
            s.close();
        }
    }
}
```

## Collections

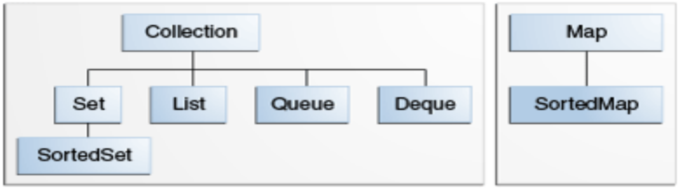
- sometimes called a container
- is simply an object that groups multiple elements into a single unit
- collections are used to store, retrieve, manipulate, and communicate aggregate data.
- https://docs.oracle.com/javase/tutorial/collections/intro/index.html

## Collections Interfaces



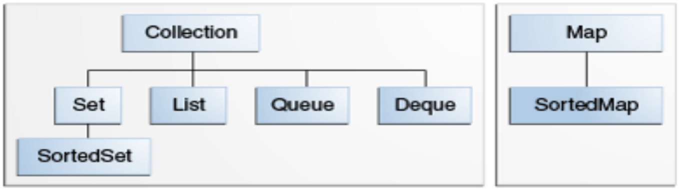
- **Collection** — the root of the collection hierarchy. A collection represents a group of objects known as its elements.

## Collections Interfaces



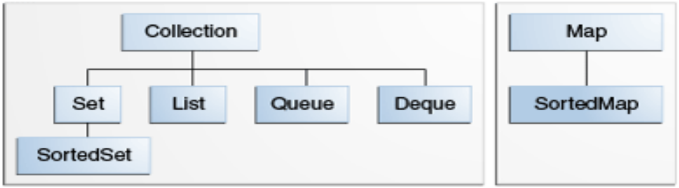
- **Set** — a collection that cannot contain duplicate elements.

## Collections Interfaces



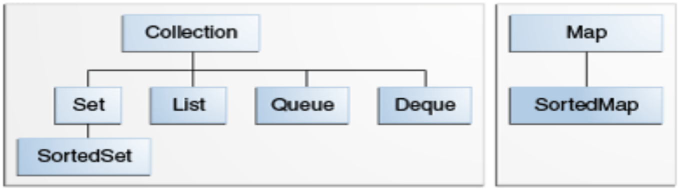
- **List** — an ordered collection (sometimes called a sequence). Lists can contain duplicate elements.

## Collections Interfaces



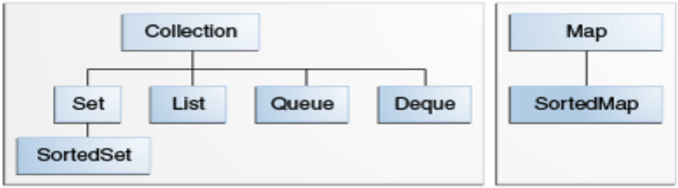
- **Queue** — a collection used to hold multiple elements prior to processing.

## Collections Interfaces



- **Dequeue** — provides additional insertion, extraction, and inspection operations.

## Collections Interfaces



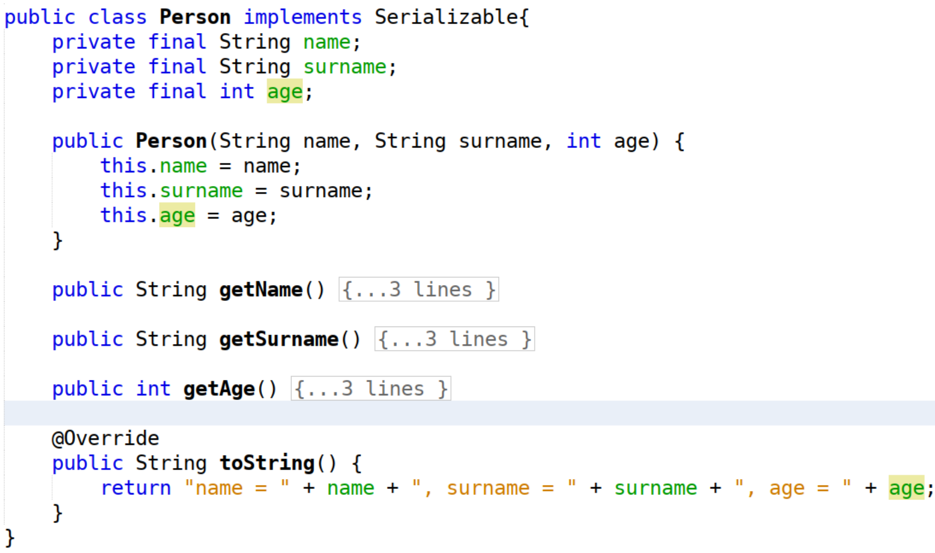
- **Map** — an object that maps keys to values.

- Interface Collection<E>
- https://docs.oracle.com/javase/8/docs/api/

## Exercise

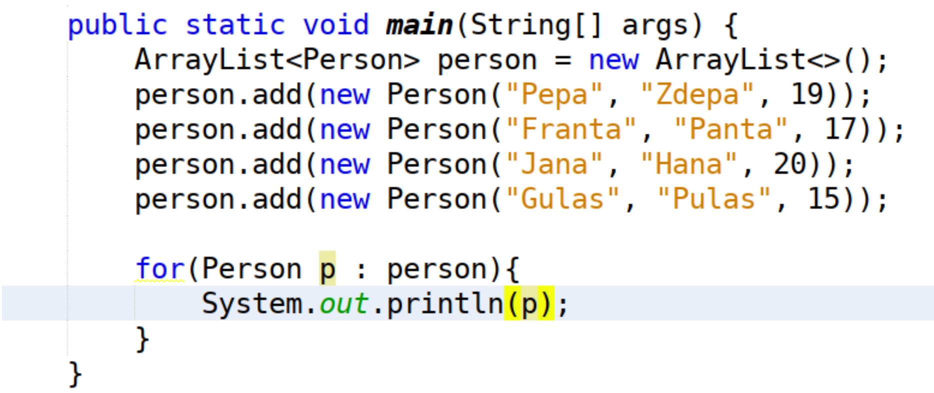
- Create a class Time, which contains "hours, minutes and seconds".
- Create a class Person, which contains "name, surname, and LinkedList<Time>".
- Create a method getName() in class Person, which returns the connected name and surname.
- Create a class Loader, which loads the data from the file into an ArrayList<Person>. Data file contains name, surname, and several values of times in the form hh:mm,ss.
- Create a method getBestTime() in class Person, which returns the object with the best time.
- Print into a file using methods getName() and getBestTime() all items in the form name, surname, bestTime.

# Advanced Collections



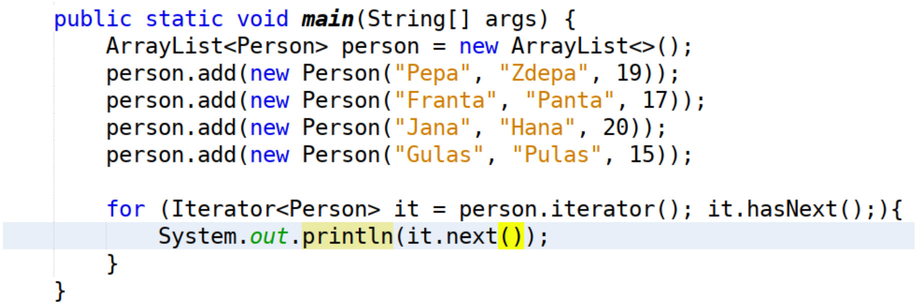
```java
public class Person implements Serializable{
    private final String name;
    private final String surname;
    private final int age;

    public Person(String name, String surname, int age) {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }

    public String getName() {...3 lines }

    public String getSurname() {...3 lines }

    public int getAge() {...3 lines }

    @Override
    public String toString() {
        return "name = " + name + ", surname = " + surname + ", age = " + age;
    }
}
```

## foreach

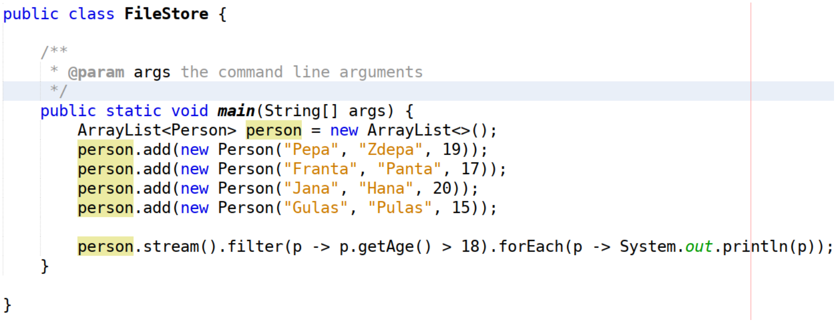


```java
public static void main(String[] args) {
    ArrayList<Person> person = new ArrayList<>();
    person.add(new Person("Pepa", "Zdepa", 19));
    person.add(new Person("Franta", "Panta", 17));
    person.add(new Person("Jana", "Hana", 20));
    person.add(new Person("Gulas", "Pulas", 15));

    for(Person p : person){
        System.out.println(p);
    }
}
```

## Iterator

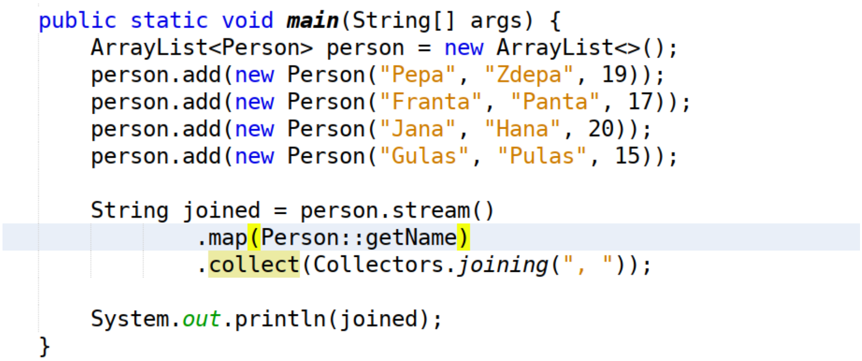

```
public static void main(String[] args) {
    ArrayList<Person> person = new ArrayList<>();
    person.add(new Person("Pepa", "Zdepa", 19));
    person.add(new Person("Franta", "Panta", 17));
    person.add(new Person("Jana", "Hana", 20));
    person.add(new Person("Gulas", "Pulas", 15));

    for (Iterator<Person> it = person.iterator(); it.hasNext();){
        System.out.println(it.next());
    }
}
```

## Aggregate Operations



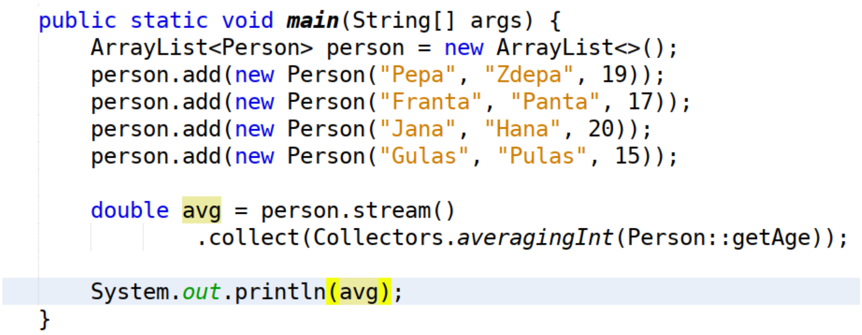
```
public class FileStore {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        ArrayList<Person> person = new ArrayList<>();
        person.add(new Person("Pepa", "Zdepa", 19));
        person.add(new Person("Franta", "Panta", 17));
        person.add(new Person("Jana", "Hana", 20));
        person.add(new Person("Gulas", "Pulas", 15));

        person.stream().filter(p -> p.getAge() > 18).forEach(p -> System.out.println(p));
    }

}
```

## Aggregate Operations



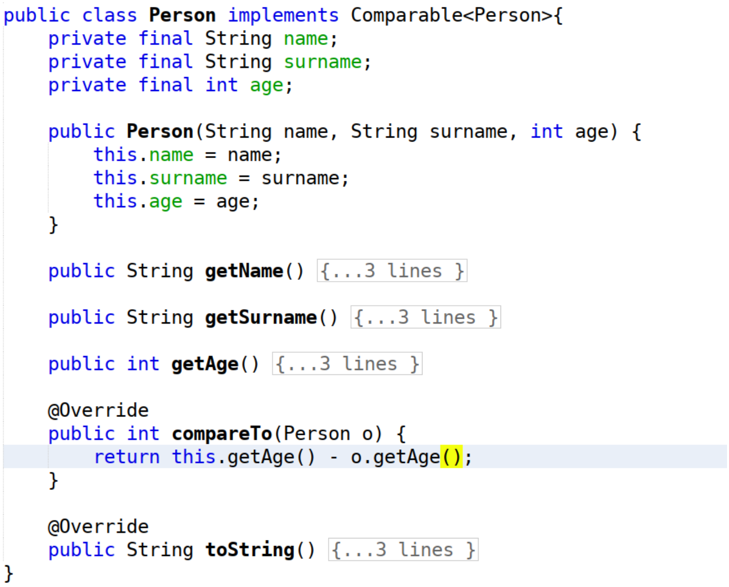
```
public static void main(String[] args) {
    ArrayList<Person> person = new ArrayList<>();
    person.add(new Person("Pepa", "Zdepa", 19));
    person.add(new Person("Franta", "Panta", 17));
    person.add(new Person("Jana", "Hana", 20));
    person.add(new Person("Gulas", "Pulas", 15));

    String joined = person.stream()
            .map(Person::getName)
            .collect(Collectors.joining(", "));

    System.out.println(joined);
}
```

## Aggregate Operations



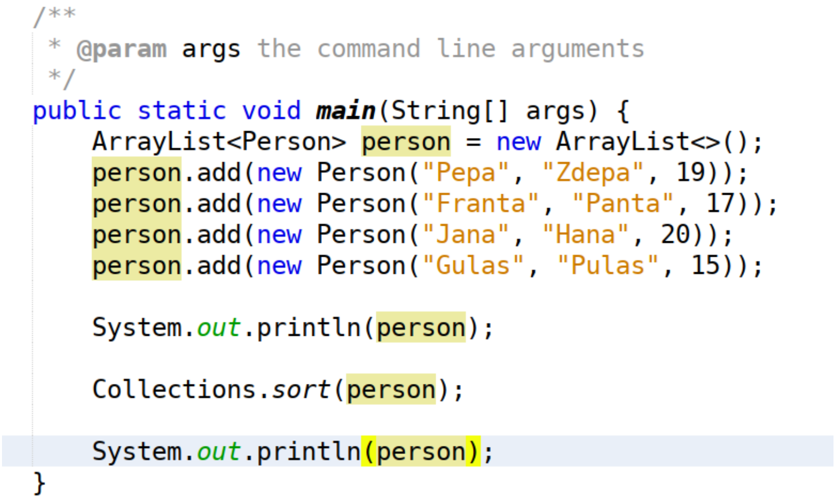
```java
public static void main(String[] args) {
    ArrayList<Person> person = new ArrayList<>();
    person.add(new Person("Pepa", "Zdepa", 19));
    person.add(new Person("Franta", "Panta", 17));
    person.add(new Person("Jana", "Hana", 20));
    person.add(new Person("Gulas", "Pulas", 15));

    double avg = person.stream()
            .collect(Collectors.averagingInt(Person::getAge));

    System.out.println(avg);
}
```

## Sorting



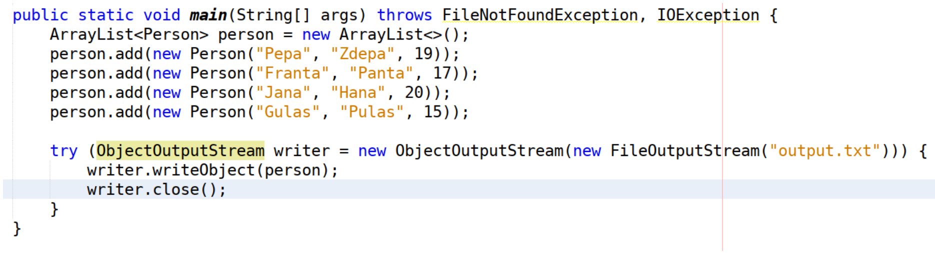
```java
public class Person implements Comparable<Person>{
    private final String name;
    private final String surname;
    private final int age;

    public Person(String name, String surname, int age) {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }

    public String getName() {...3 lines }

    public String getSurname() {...3 lines }

    public int getAge() {...3 lines }

    @Override
    public int compareTo(Person o) {
        return this.getAge() - o.getAge();
    }

    @Override
    public String toString() {...3 lines }
}
```

## Sorting



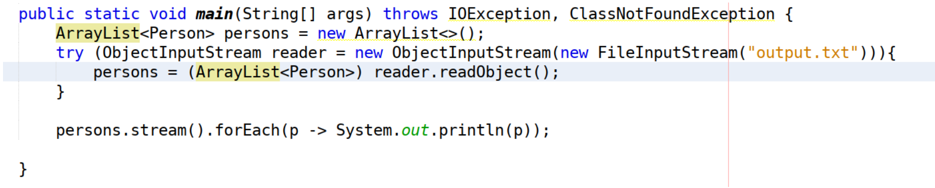
```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    ArrayList<Person> person = new ArrayList<>();
    person.add(new Person("Pepa", "Zdepa", 19));
    person.add(new Person("Franta", "Panta", 17));
    person.add(new Person("Jana", "Hana", 20));
    person.add(new Person("Gulas", "Pulas", 15));

    System.out.println(person);

    Collections.sort(person);

    System.out.println(person);
}
```

## Advanced Streams



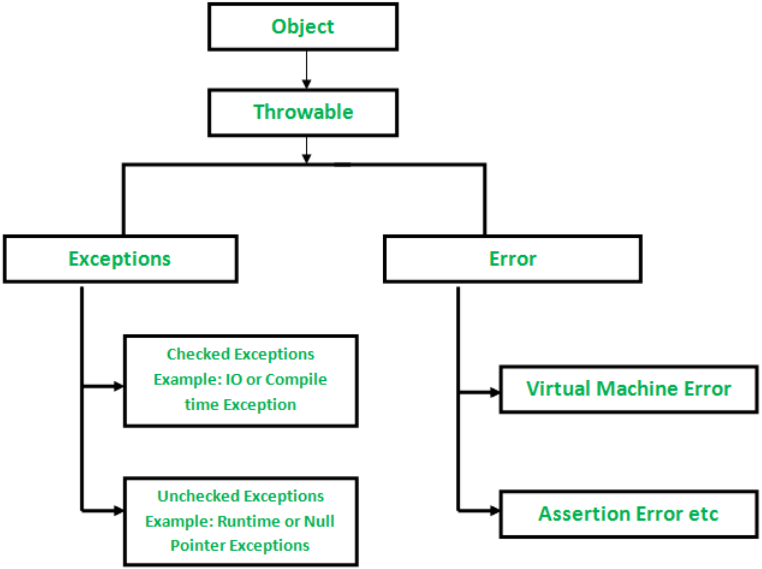
```
public static void main(String[] args) throws FileNotFoundException, IOException {
    ArrayList<Person> person = new ArrayList<>();
    person.add(new Person("Pepa", "Zdepa", 19));
    person.add(new Person("Franta", "Panta", 17));
    person.add(new Person("Jana", "Hana", 20));
    person.add(new Person("Gulas", "Pulas", 15));

    try (ObjectOutputStream writer = new ObjectOutputStream(new FileOutputStream("output.txt"))) {
        writer.writeObject(person);
        writer.close();
    }
}
```

# Advanced Streams



```java
public static void main(String[] args) throws IOException, ClassNotFoundException {
    ArrayList<Person> persons = new ArrayList<>();
    try (ObjectInputStream reader = new ObjectInputStream(new FileInputStream("output.txt"))){
        persons = (ArrayList<Person>) reader.readObject();
    }

    persons.stream().forEach(p -> System.out.println(p));

}
```

## Working with errors in JAVA


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

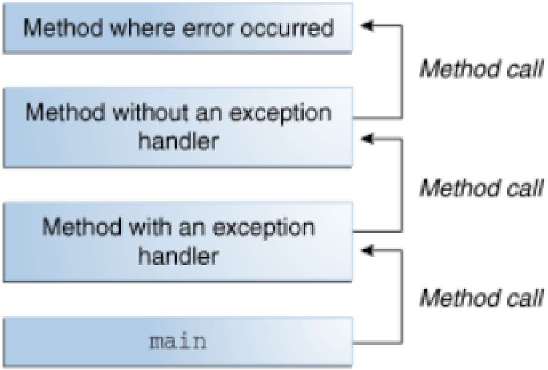
## Exceptions

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

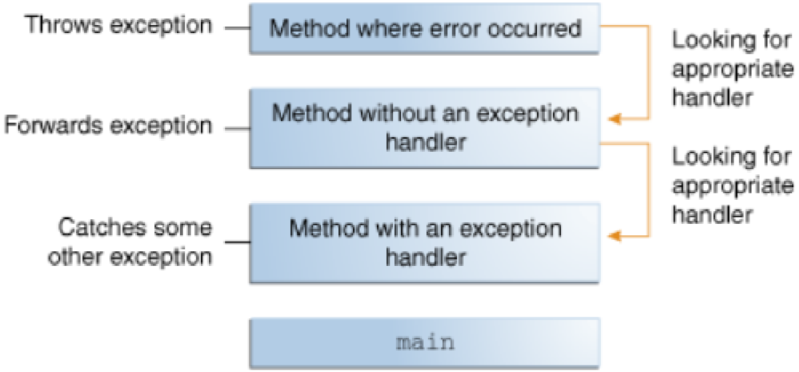
## Exceptions

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

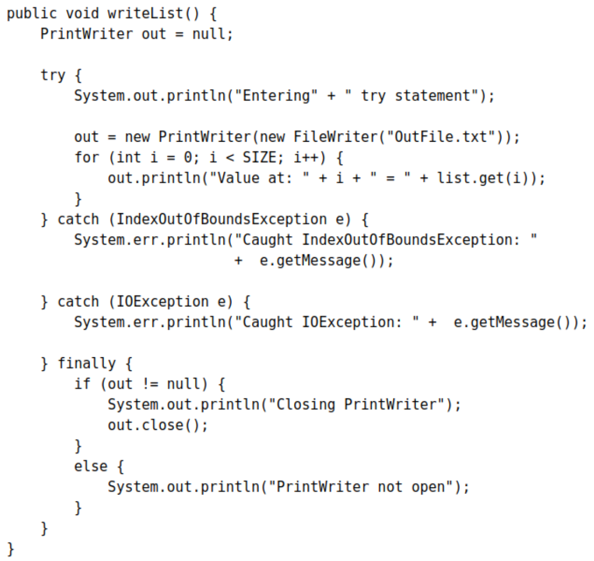
## Exceptions


Method where error occurred
Method without an exception handler
Method with an exception handler
main
Method call
Method call
Method call

## Exceptions

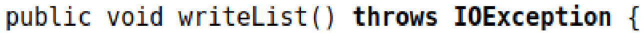

Throws exception
Method where error occurred
Looking for appropriate handler
Forwards exception
Method without an exception handler
Looking for appropriate handler
Catches some other exception
Method with an exception handler
main

## Exceptions



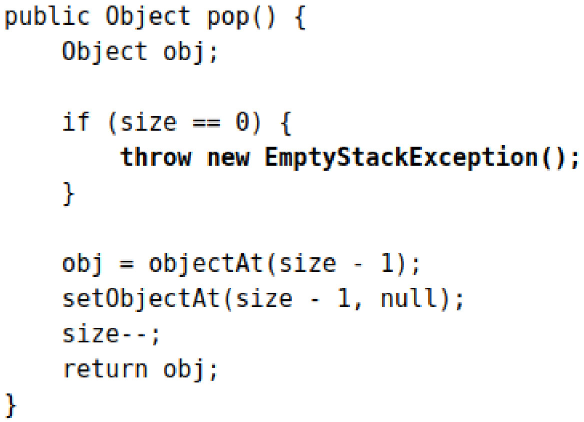
```
public void writeList() {
    PrintWriter out = null;

    try {
        System.out.println("Entering" + " try statement");

        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
    } catch (IndexOutOfBoundsException e) {
        System.err.println("Caught IndexOutOfBoundsException: "
                         +  e.getMessage());

    } catch (IOException e) {
        System.err.println("Caught IOException: " +  e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        }
        else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

## Exceptions



```
public void writeList() throws IOException {
```

## Exceptions



```
public Object pop() {
    Object obj;

    if (size == 0) {
        throw new EmptyStackException();
    }

    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

## Exercise

- Create a class Student containing name::String, surname::String, and points::int
- Create an ArrayList of Student and add there 5 students.
- Create a class Writer which store the ArrayList of Student into a file using ObjectOutputStream
- Create a class Reader which load the ArrayList of Student from a file using ObjectInputStream
- Create a class ExerciseClass which contains methods allowing print the student based on minimal point, counting average (poins) and returning the String of all names (using foreach, iterator, agragated operations)
- Sort the students using Collections.sort()

## Thank you for your attention

Lukas Tomaszek

VSB – Technical University of Ostrava

lukas.tomaszek@vsb.cz

October 31, 2019

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE