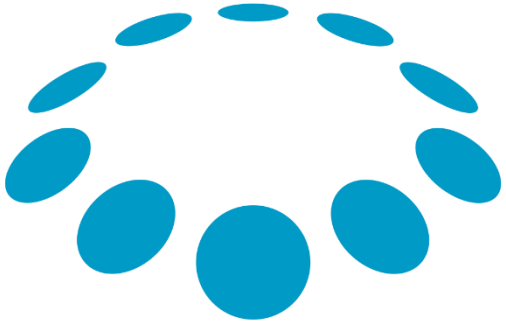
Data Streams in Java (Sockets, Collection Data)

Dr. Eliahu Khalastchi

2018

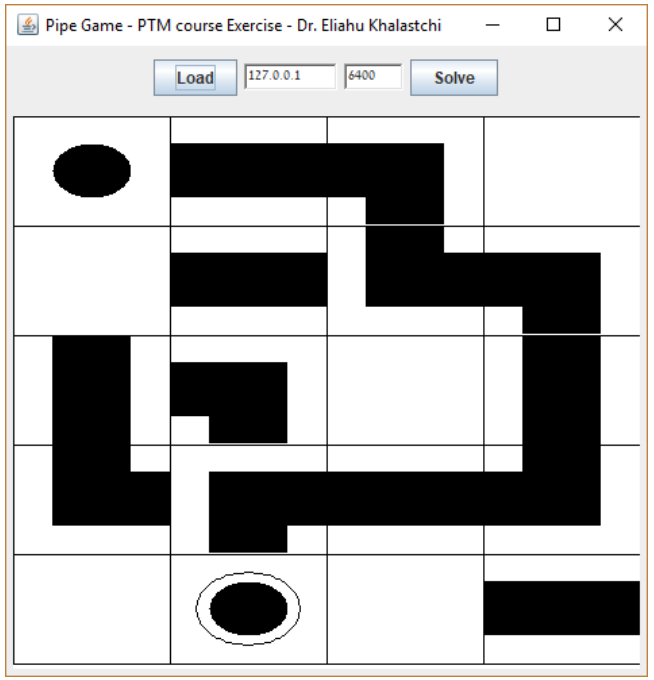


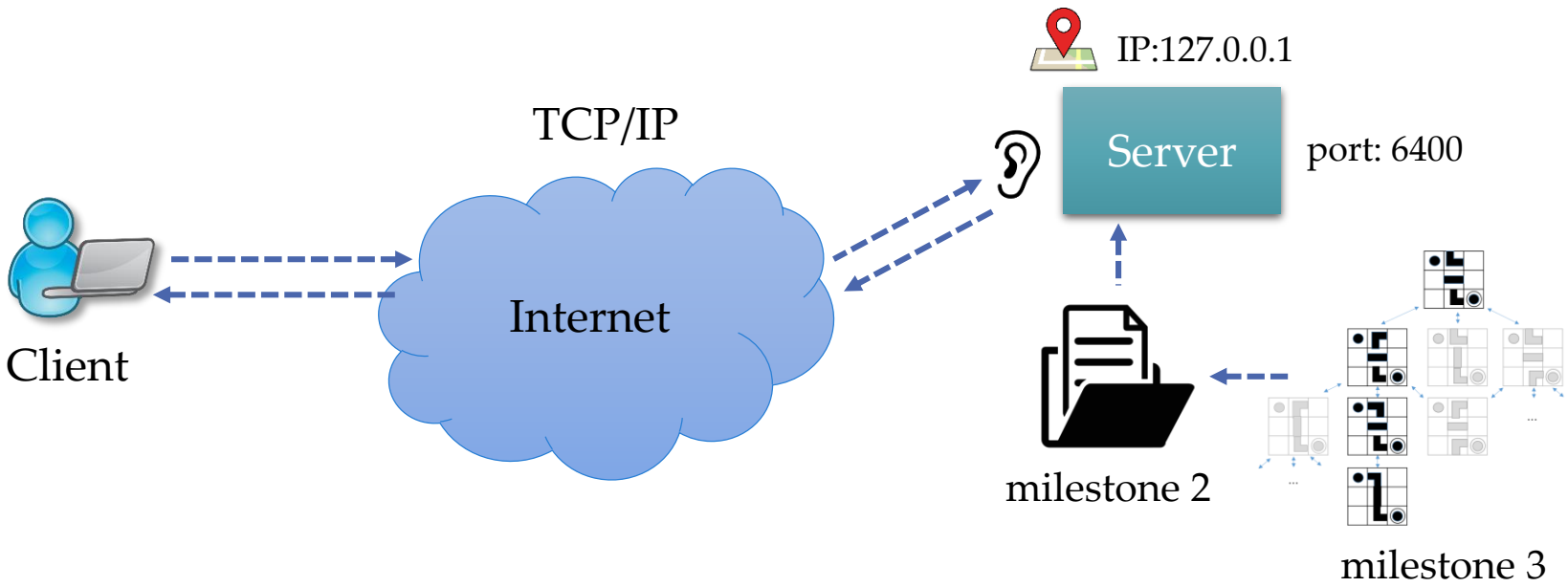
המסלול האקדמי
המכללה למינהל

Streaming with sockets!

Basic API…

Our goal…


Pipe Game - PTM course Exercise - Dr. Eliahu Khalastchi
Load
127.0.0.1
6400
Solve


IP:127.0.0.1
TCP/IP
Server
port: 6400
Client
Internet
milestone 2
milestone 3

Command Line Interface - TCP/IP Client

Let's build a method that gets the user's input and sends it somewhere

- We don't want to define the source and destination
  - this should be parameterized

```
public class CLIclient {
                                                    Any source      Any destination   Protocol dependent
  private void readInputsAndSend(BufferedReader in, PrintWriter out,String exitStr){
    try {
      String line;
      while(!(line=in.readLine()).equals(exitStr)){
        out.println(line);
        out.flush();
      }
    } catch (IOException e) { e.printStackTrace();}
  }
```

```
public void start(String ip, int port){
  try {
    Socket theServer=new Socket(ip, port);
    System.out.println("connected to server");

    BufferedReader userInput=new BufferedReader(new InputStreamReader(System.in));
    BufferedReader serverInput=new BufferedReader(new InputStreamReader(theServer.getInputStream()));

    PrintWriter outToServer=new PrintWriter(theServer.getOutputStream());
    PrintWriter outToScreen=new PrintWriter(System.out);

    // correspond according to a well-defined protocol
    readInputsAndSend(userInput,outToServer,"exit");
    readInputsAndSend(serverInput,outToScreen,"bye");

    userInput.close();
    serverInput.close();
    outToServer.close();
    outToScreen.close();
    theServer.close();

  } catch (UnknownHostException e) {/*...*/}
    catch (IOException e) {/*...*/}
}
```

```
public static void main(String[] args) {
  String ip=args[0];
  int port = Integer.parseInt(args[1]);
  CLIclient client=new CLIclient();
  client.start(ip, port);
}
```

Server Example

TCP/IP

Basic API – handle 1 client

```
ServerSocket server=new ServerSocket(port);
server.setSoTimeout(1000);
try{
 Socket aClient=server.accept(); // blocking call

 InputStream inFromClient=aClient.getInputStream();
 OutputStream outToClient=aClient.getOutputStream();

 // interact (read & write) with the client according to protocol

 inFromClient.close();
 outToClient.close();
 aClient.close();
 server.close();
}catch (SocketTimeoutException e) {/*...*/}
```

Loop this

and be able to stop

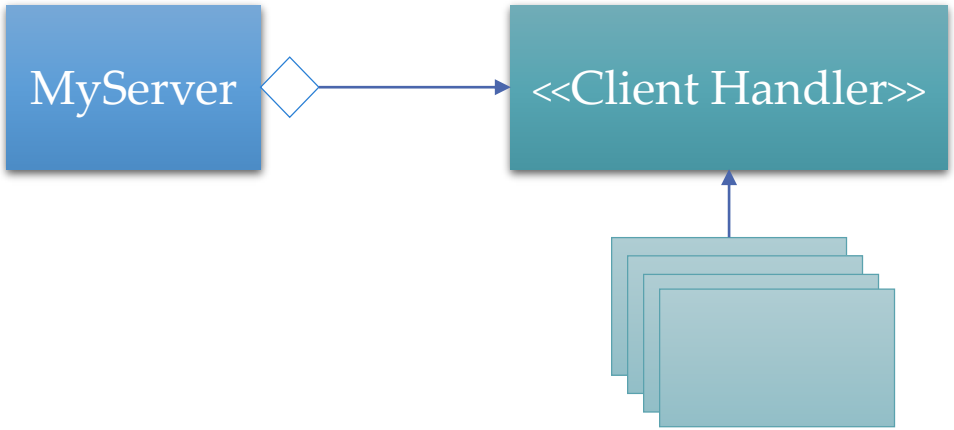
We want to delegate this

Thread this

Let's delegate via strategy pattern

```
public interface ClientHandler{
  void handleClient(InputStream inFromClient, OutputStream outToClient);
}
```

```
public class MyServer {

  private int port;
  private ClientHandler ch;
  private volatile boolean stop;

  public MyServer(int port,ClientHandler ch) {
    this.port=port;
    this.ch=ch;
    stop=false;
  }
```



Let's loop and handle multiple clients

```
private void runServer()throws Exception {
  ServerSocket server=new ServerSocket(port);
  server.setSoTimeout(1000);
  while(!stop){
    try{
      Socket aClient=server.accept(); // blocking call
        try {

          ch.handleClient(aClient.getInputStream(), aClient.getOutputStream());

          aClient.getInputStream().close();
          aClient.getOutputStream().close();
          aClient.close();

        } catch (IOException e) {/*...*/}
    }catch(SocketTimeoutException e) {/*...*/}
  }
  server.close();
}
```

Let's enable stopping the server

```
public void start(){
   runServer();
}

public void stop(){
   stop=true;
}
```

```
// in main()
MyServer server=new MyServer(port, myClientHandler);
server.start(); // should be in a different thread…
// ... wait for administrator to close
server.stop();
```

Let's enable stopping the server

```
public void start(){
  new Thread(()->runServer()).start(); // we will learn about it semester B…
}

public void stop(){
  stop=true;
}
```

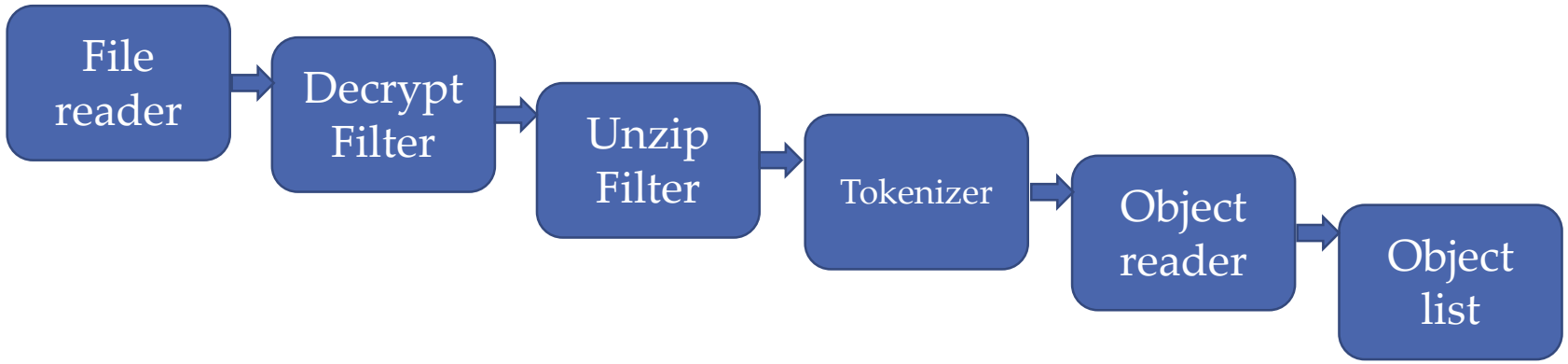
```
// in main()
MyServer server=new MyServer(port, myClientHandler);
server.start(); // should be in a different thread…
// ... wait for administrator to close
server.stop();
```

Collection API - Streams

New in java 8

Pipes and Filters Architecture

- The concept: a data stream is passed by pipes through filters
- Each filter manipulates the data stream and passes it on to the next filter
- Example:



Stream

- Represents a **sequence of elements** form a given source
- **Does not store** elements, but rather compute them on **demand**
- Possible sources
  - Collections, generating function, I/O, etc…
- **Pipelined** usage (fluent)
- **Intermediate** operations / **terminal** operations
- **Lazy:** only terminal operations trigger a computation

Filter

```
List<Point> data=new LinkedList<>();
Random r=new Random();
for(int i=0;i<100;i++)
  data.add(new Point(r.nextInt(200)-100,r.nextInt(200)-100));

data.stream().filter(p->p.getX()>0).forEach(System.out::println);
```

A terminal operation

default Stream<E> stream()

Returns a sequential Stream with this collection as its source.

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

Returns a stream consisting of the elements of this stream that match the given predicate.
This is an **intermediate operation**.

Stream

Intermediate

returns a Stream

- distinct()
- map()
- flatMap()
- limit()
- peek()
- sorted()

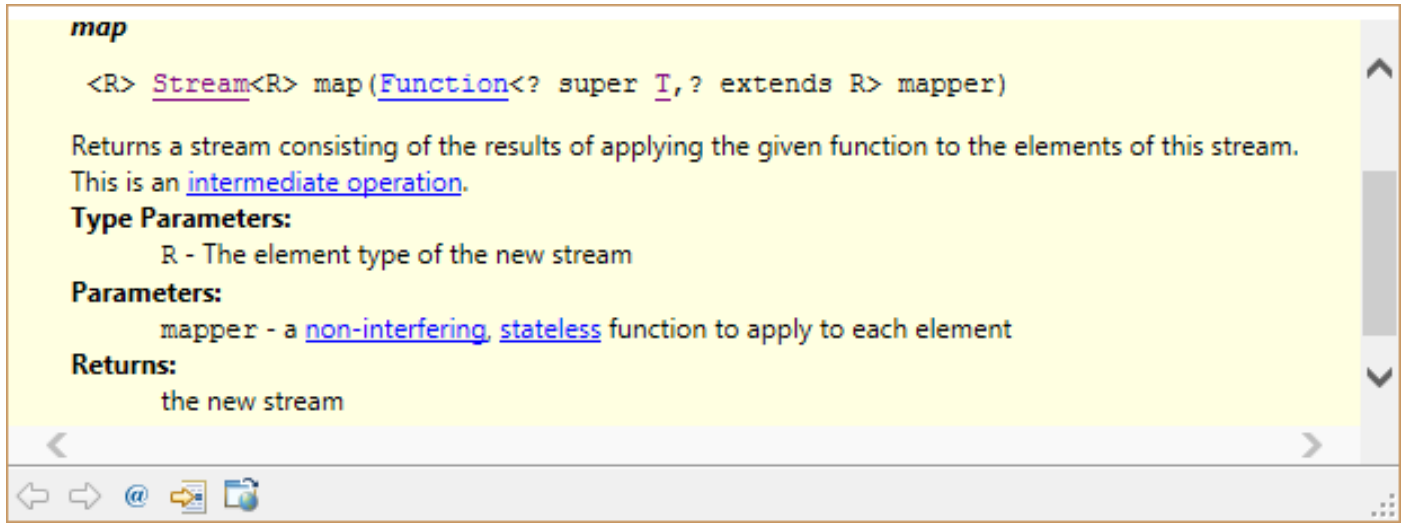
Terminal

returns a result

- collect()
- count()
- forEach()
- min() , max()
- reduce()
- toArray()
- findAny() , findFirst()
- allMatch() , andMatch() , noneMatch()

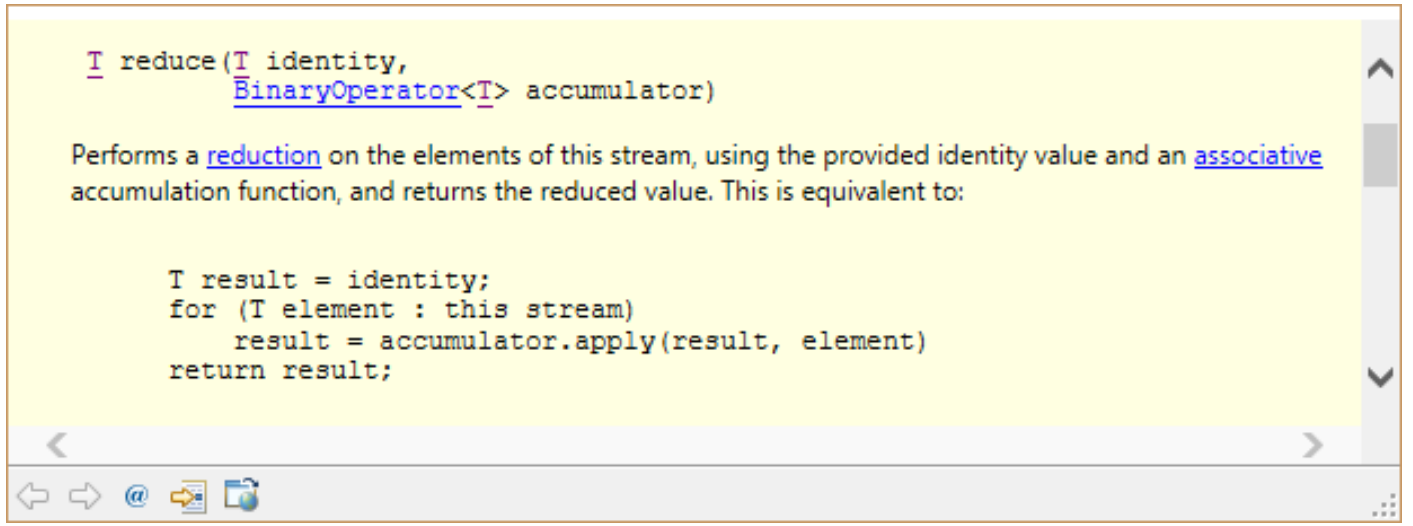
Map-Reduce Example

```
List<String> strings=Arrays.asList("the", "answer", "to", "life", "the", "universe",
"and", "everything", "=", "42");

int totalLength = strings.stream().map(String::length).reduce(0, (x,y)->x+y);
System.out.println(totalLength); // wow! its 42!!
```



Map-Reduce Example

```
List<String> strings=Arrays.asList("the", "answer", "to", "life", "the", "universe",
"and", "everything", "=", "42");

int totalLength = strings.stream().map(String::length).reduce(0, (x,y)->x+y);
System.out.println(totalLength); // wow! its 42!!
```



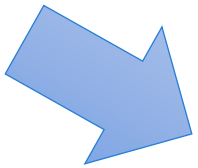
```
T reduce(T identity,
         BinaryOperator<T> accumulator)
```

Performs a reduction on the elements of this stream, using the provided identity value and an associative accumulation function, and returns the reduced value. This is equivalent to:

```
T result = identity;
for (T element : this stream)
    result = accumulator.apply(result, element)
return result;
```

Filtering info from Files

```
BufferedReader in=new BufferedReader(new FileReader(fileName));
List<String> errors=new LinkedList<>();
String line;
while(errors.size()<10 && (line=in.readLine())!=null)
  if(line.startsWith("ERROR:"))
    errors.add(line);

in.close();
```



also new to Java 8

```
errors=Files.lines(Paths.get(fileName))
  .filter(s->s.startsWith("ERROR:"))
  .limit(10)
  .collect(Collectors.toList());
```

groupinBy

```
List<Employee> employees=new LinkedList<>();
employees.add(new Employee(18, "Moshe"));
employees.add(new Employee(18, "Tzipi"));
employees.add(new Employee(25, "Alon"));
employees.add(new Employee(22, "Tal"));
employees.add(new Employee(22, "Tomer"));

Map<Integer,List<Employee>> EmpByAge = employees.stream()
        .filter(e->e.name.startsWith("T"))
        .collect(Collectors.groupingBy(e->e.age));

EmpByAge.forEach((age,emps)->{
        System.out.println(age+":");
        emps.forEach(e->System.out.println("\t"+e.name));
});
```

```
output:
18:
        Tzipi
22:
        Tal
        Tomer
```

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html

Parallel Streams

fork-join pool

```
long sum = employees.parallelStream()
        .filter(e-> e.getClass().equals(Manager.class))
        .map(Employee::getSalary)
        .reduce(0, (x,y)->(x+y));
```