Cursus Robbe Magerman Object Oriënted Software Development 2

Robbe Magerman

15/05/2024

Inhoudstafel

1 Hoofdstuk 1

1.1 Abstracte classes

1.1.1 Wat?

`public abstract class Rekening`

- Eenmaal dit geïmplementeerd moeten alle kinderen die van deze klasse `extends` die methodes zeker gebruiken.

2 Hoofdstuk 2

2.1 Interfaces

2.1.1 Wat?

`public interface Beheerstkost`

- Eenmaal dit geïmplementeeerd **moeten* alle kinderen die van deze klasse `implements` de gegegeven methodes implementeren via een `@Override`

3 Hoofdstuk 3 Lamda Expressies

3.1 Comparators - CompareTo

3.1.1 Comparator Algemeen

- `class Movie implements Comparable<Movie>`
- `hashcode` en `equals` genereren

3.1.2 Comparator via een String

```
@Override
public int compareTo(Movie m) {
    int compareName = name.compareTo(m.name);
    return compareName != 0 ? compareName : year - m.year;
}
```

3.1.3 Comparator via een int

```
@Override
    public int compareTo(Container c) {
        return Integer.compare(this.serialNumber, c.serialNumber);
    }d
```

3.1.4 Comparator via eerst een int daarna een String

```
@Override
    public int compareTo(Auto a) {
        int result = Integer.compare(aantalOnderhoudsbeurten, a.
           aantalOnderhoudsbeurten);
        if (result != 0)
            return result;
        return nummerplaat.compareTo(a.nummerplaat);
    }
```

3.2 Comparators - Compare

3.2.1 Comparators in een andere klasse - Algemeen

- We **implementeren** de klasse waarin we gaan comparen:
  - `public class MassacComparator implements Comparator<Container>`

3.2.2 Comparators in een andere klasse via een int

```
@Override
    public int compare(Container c1, Container c2) {
        return Integer.compare(c1.getMassa(), c2.getMassa());
    }
```

3.2.3 Comparators in een andere klasse via een String

```
@Override
    public int compare(Container c1, Container c2) {
        return c1.getEigenaar().compareTo(c2.getEigenaar());
    }
```

3.2.4 Comparators in een andere klasse via een volume, daarna een String

```
@Override
    public int compare(Container c1, Container c2) {
        int volumeCompare = Integer.compare(c1.getVolume(), c2.getVolume());
        if (volumeCompare != 0)
            return volumeCompare;
        return c1.getEigenaar().compareTo(c2.getEigenaar());
    }
```

3.3 Lamda Expressies

3.3.1 Comparen met een merk daarna een model (String en String)

```
Collections.sort(autos, Comparator.comparing(Auto::getMerk).thenComparing(Auto
   ::getModel));
```

3.3.2 Comparen via een merk (String)

```
Collections.sort(autos, Comparator.comparing(Auto::getMerk));
```

3.4 Comparator - Anonieme innerklasse

3.4.1 Een compleet nieuwe Comparator via een int

```
Comparator<Movie> yearComp1 = new Comparator<Movie>() {
            @Override
            public int compare(Movie m1, Movie m2) {
                return Integer.compare(m1.getYear(), m2.getYear());
            }
        };
```

3.4.2 Een comparator aan de hand van een domeinklasse

```
Collections.sort(allMovies, new YearComparator());
        showMovies("Movies with total ordering based on year", allMovies);
```

4 Hoofdstuk 4: Exceptions en robuustheid

4.1 Volledig Robuust menu

4.1.1 Header van een Menu

```
    public int kiesUitMenu() {
        String foutmelding = String.format("%n === Gelieve een getal tussen 1
            en 3 in te voeren...%nsdf ===");
        boolean geldigeInvoer = false;
        int keuze = -1;
        do {
            try {
                System.out.printf("1. Toon een overzicht van alle producten %n2
                    . Voeg een plant Toe%n3. Afsluiten%n");
                System.out.printf("Maak je keuze: ");
                keuze = Integer.parseInt(invoer.nextLine());
                geldigeInvoer = keuze >= 1 && keuze <= 3;
                if (!geldigeInvoer) {
                    System.out.println(foutmelding);
                }
            } catch (NumberFormatException e) {
                System.out.printf(foutmelding);
            } catch (IllegalArgumentException e) {
                System.out.println(" === Er ging nog iets harder fout ===");
            }
        } while (!geldigeInvoer);
        return keuze;
    }
```

4.1.2 Robuust Menu via een double

```
    public double geefPrijsPlant() {
        String foutmelding = "\n === Geef een geldig nummer in ===";
        boolean geldigeInvoer = false;
        double prijs = 0;
        do {
            try {
                System.out.println("Geef de prijs van de plant op: ");
                prijs = Double.parseDouble(invoer.next());
                geldigeInvoer = prijs > 0;
                if (!geldigeInvoer)
                    System.out.printf("%s", foutmelding);
            } catch (NumberFormatException e) {
                System.out.println("\n === Dit is geen getal ===");
            } catch (IllegalArgumentException e) {
                System.out.println("\n === Iets ging fout in verband met al de
                    rest ===");
            }
        } while (!geldigeInvoer);
        return prijs;
    }
```

4.1.3 Robuust Menu via een String

```
    private String geefNaamPlant() {
```

```
        String naam = "";
        String foutmelding = ("\n === Geef een geldige gebruikersnaam in ===");
        boolean geldigeInvoer = false;
        do {
            try {
                System.out.print("Geef de naam van de plant in: ");
                naam = invoer.nextLine();
                // groter dan 2, niet leeg
                geldigeInvoer = naam.length() > 2 && !naam.isBlank() && !naam.
                    isEmpty();
                if (!geldigeInvoer)
                    System.out.println(foutmelding);
            } catch (IllegalArgumentException e) {
                System.out.println(foutmelding);
            }
        } while (!geldigeInvoer);
        return naam;
    }
```

5 Hoofdstuk 5: JavaFX

5.1 Hoe krijg je een startscherm:

- Maak een nieuw project aan
- Maak de `StartUp` klasse aan in `package main` en laat het *erven* van Application.

5.2 JavaFX codes

5.2.1 Button

```
buttonAfsluiten = new Button("afsluiten");
```

5.2.2 Css toevoegen

```
buttonAfsluiten.setStyle("-fx-background-color: red;");
```

5.2.3 Font toevoegen

```
// font toevoegen als bestand
Button buttonAfsluiten.setFont(Font.font("Berkshire Swash", 15));
```

5.2.4 Action op een button + afsluiten + alert

```
buttonAfsluiten.setOnAction((evt) -> {
            Alert alertAfsluiten = new Alert(AlertType.INFORMATION, "Bedankt
               voor het spelen, tot de volgende keer!");
            alertAfsluiten.showAndWait();
            Platform.exit();
        });
```

5.2.5 Image

```
ImageView imageViewFavicon = new ImageView();
        Image imageFavicon = new Image(getClass().getResourceAsStream("/gui/img
           /mascotte_rechtgedraaid.png"));
        imageViewFavicon.setImage(imageFavicon);
        this.getChildren().addAll(imageViewFavicon);
```

5.2.6 Switchen van scherm

```
getScene().setRoot(registratieScherm);
```

5.2.7 Label

```
Label label = new Label("Hier zet ik tekst op");
```

5.2.8 Hyperlink

```
Hyperlink linkForgot = new Hyperlink("Forgot password");
```

5.2.9 Padding

```
Insets padding = new Insets(10);
label.setMargin(padding);
```

5.2.10 Spacing

```
setSpacing(10);
```

5.3 JavaFX Lay-outcontainers

Box	Betekenis
`VBox`	Plaats de inhoud op een *verticale* as op elkaar
`HBox`	Plaats de inhoud op een *horizontale* as op elkaar
`GridPane`	Geeft de inhoud elk een *specifieke plaats*
`BorderPane`	Plaats de inhoud aan de *randen* van het scherm of in het midden
`Stackpane`	Plaatst de inhoud *boven elkaar* plaatst en zichzelf aanpast aan de grootte van zijn kinderen. Zo maak je bv een achtergrond op een scherm
`AnchorPane`	Plaatst de inhoud ten opzichte van de *randen* van de container of ten opzichte van elkaar door ankers te gebruiken.
`FlowPanes`	Plaatst de inhoud in een *flow van links naar rechts* en van boven naar onder. Het past automatisch de grootte van zijn kinderen aan en kan over meerdere rijen en kolommen worden verdeeld.
`TilePane`	Plaatst de inhoud in een *tegelpatroon*. Het past automatisch de grootte van zijn kinderen aan en kan worden geconfigureerd om zijn kinderen te ordenen in een vaste rij of kolom, of om ze vrij te laten stromen.

6 Hoofdstuk 6: Collecties

6.1 Structuren

6.1.1 Theorie

- Een List is een `Collection` het returntype vanaf nu dat de DomeinController dus zal ontvangen is een `Collection` , hier zullen dus enkel methodes worden toegepaste van de `interface Collection`

6.1.2 ArrayAsList

- Dit werkt niet als een gewone (dynamische) array waarbij je dingen kan toevoegen

```
private final List<Stripfiguur> stripfiguren;

stripfiguren = new ArrayList<>(Arrays.asList(mapper.geefStripfiguren()));
```

6.1.3 Collections.sort()

- eerst maken we een hashcode en equals in de `domeinklasse`
- Daarna maken we een compare en typen we dit in de `repository`

```
@Override
public int compareTo(Stripfiguur f) {
    int naamComparison = naam.compareTo(f.naam);
    return naamComparison == 0 ? Double.compare(grootte, f.grootte) :
       naamComparison;
}
```

- Waarbij we daarna dit implementeren in de Repository

```
public Collection<Stripfiguur> geefStripfigurenGesorteerdOpNaam() {
    Collections.sort(stripfiguren);
    return stripfiguren;
}
```

- We kunnen deze klasse ook uitbreiden met een lamda expressie:

```
public Collection<Stripfiguur> geefStripfigurenGesorteerdOpNaam() {
    Collections.sort(stripfiguren, Comparator.comparing(Stripfiguur::
       getNaam).reversed());
    return stripfiguren;
}
```

- alle stripfiguren zonder dubbels

```
public Collection<Stripfiguur> geefStripfigurenZonderDubbels() {
    Collection<Stripfiguur> stripfigurenZonderDubbels = new HashSet<>(
       stripfiguren);
    return stripfigurenZonderDubbels;
}
```

- Drie willekeurige stripfiguren

```
public Collection<Stripfiguur> geefDrieWillekeurigeStripfiguren() {
    Collections.shuffle(stripfiguren);
    return stripfiguren.subList(0, 3);
}
```

- Iets toevoegen aan een lisjt

```
    public void voegStripfiguurToe(String naam, double grootte) {
        stripfiguren.add(new Stripfiguur(naam, grootte));
    }
```

- Overstappen van array naar Collection

```
// normale methode
    stripfiguren.addAll(Arrays.asList(mapper.geefExtraStripfiguren()));

// stripfiguren toevoegen zonder dubbels3+
    HashSet<>(Arrays.asList(mapper.geefExtraStripfiguren()));
```

7 Hoofdstuk 7:

7.1 imperatief en declaratief

7.1.1 imperatief (forlus in java)

Hoe en wat, explicitate iteratie

```
for (Employee e : employees){
    if (e.getSalary() > 10000)
        richEmployeesNames.add(e.getName());
}
```

7.1.2 declaratief (select in sql)

Wat, geen explicitie iteratie

```
select name
from eployees
where salary > 10000
```

7.1.3 Interface Stream

- Een Stream is een `sequentie` van objecten waarop geaggrgeerde en parallelle operaties kunnen toegepast worden
- We slaan er geen data in op
- We gebruiken het om iets te doen op de data
- Stap 1: we vormen een `Collection` / `Array` naar een Stream
- Stap 2: toevoegen van operaties die opnieuw een Stream opleveren (mapper, filteren . . .) we noemen dit ook intermediate opertations
  - Dit heeft als invoer een Stream en uitvoer ook
  - Bij een Terminal operation gaan we uit de Stream
- Stap 3: We gaan uit de Stream wereld en komen weer terug in de Java wereld

7.1.4 . . . Stream().of()

- Je moet kijken naar het type waarop je een stream toepast
  - int? –> intStream().of()
  - Objecten? –> stream().of()

7.1.5 Arrays.stream && Arrays.asList TRèS IMPORTANT

- Overstappen van Array naar Streams
- Overstappen van Array naar Collection

7.1.6 verschil tussen map en mapTO. . .

```
.map(e -> getSalary())
    Stream<Double>

.mapToDouble(e -> e.getSalary())
    DoubleStream
```

7.1.7 Filter

- Data die aan een bepaalde voorwaarde doet

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).filter(i -> i % 2 == 0);
```

7.1.8 map

-

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).map(i -> i / 2);
```

7.1.9 distinct

- Verwijdert alle dubbele waarden, (gebruikt de equals methode in de achtergrond - niet hetzelfde als Comparator)

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).distinct(.).filter(i -> i % 2 == 0);
```

7.1.10 Sorted

- Zorgt ervoor dat de uitkomst van de waarden gesorteerd is

```
int[] values = {3, 4, 6, 1 }
IntStream stream= IntStream.of(values).sorted();
```

7.1.11 OptionalInt

- Dit bevat mogelijk een waarde
- Deze klasse wordt gebruikt bij de max() functie

```
- int getAsInt() // Double, Long
- int orElse(int other)
- boolean isPresent()
```

7.1.12 findFirst()

- Dit wordt gebruikt bij streams
- Wanneer dit wordt gebruikt, wordt het 1e 'object' dat aan die voorwaarde voldoet weergegeven

7.1.13 reduce()

- Hier wordt een berekening teruggegeven
- Er is geen previousValue

```
.reduce(x, y) -> x + y
// of
.reduce (0, (x, y) -> x + y)
```

7.1.14 Collectors.toCollections

- Hier kan je alles inzetten wat je kent van Collections
- bv.

```
.collect(Collecters.toCollections(HashSet::new))
```

7.1.15 anyMatch()

- Werkt als de filtermethode, kijkt of iets aan een bepaalde voorwaarde voldoet en levert *automatisch* true of false op

7.1.16 Elementen verzamelen in een Collection

```
public Collection<Vliegmaatschappij> geefMaatschappijenMetPartners(int
    minAantal) {
     return maatschappijen.stream().filter(vm -> vm.getPartners().size() >=
        minAantal)
             .collect(Collectors.toCollection(ArrayList::new));
}
```

7.1.17 elementen alfabatisch gesorteerd zonder natuurlijke ordening

```
public Collection<Vliegmaatschappij> geefAirlinesAlfabetischGesorteerd() {
     return maatschappijen.stream().sorted().collect(Collectors.toCollection
        (ArrayList::new));
}
```

7.1.18 Elementen omzetten naar een DTO

```
private Collection<VliegmaatschappijDTO> geefVliegmaatschappijDTOs(
    Collection<Vliegmaatschappij> maatschappijen) {
     return maatschappijen.stream().map(vm -> new VliegmaatschappijDTO(vm))
             .collect(Collectors.toCollection(ArrayList::new));
}
// of
private Collection<VliegmaatschappijDTO> geefVliegmaatschappijDTOs(
    Collection<Vliegmaatschappij> maatschappijen) {
return maatschappijen.stream().map(vm -> VliegmaatschappijDTO(vm)::new)
         .collect(Collectors.toCollection(ArrayList::new));
}
```

7.1.19 Alle DTO-objecten aan elkaar koppelen in de ui

```
private String geefAlleDTOsInEenString(Collection<VliegmaatschappijDTO>
    vmDTOs) {
     return vmDTOs.stream().map(vm -> dtoToString(vm)).collect(Collectors.
        joining());
}
```

7.1.20 Gestorteerd op aantal partners

```
public Collection<Vliegmaatschappij>
    geefAirlinesGesorteerdVolgensAantalPartners() {
     return maatschappijen.stream().sorted(Comparator.comparing((
        Vliegmaatschappij v) -> v.getPartners().size())
             .thenComparing(Vliegmaatschappij::getNaam)).collect(Collectors.
                toCollection(ArrayList::new));

}
```

7.1.21 Iets teruggeven startend met ...

```
    public String geefEersteAirlineStartendMet(String startTekst) {
        return maatschappijen.stream().filter(vm -> vm.getNaam().startsWith(
            startTekst)).findFirst()
                 .map(vm -> vm.getNaam()).orElse("niet bestaande");
    }
```

7.1.22 Elementen met een aantal ...

```
    public Vliegmaatschappij geefEenAirlineMetPartner(String partner) {
        return maatschappijen.stream().filter(vm -> vm.getNaam().contains(
            partner)).findAny().orElse(null);
    }
```

7.1.23 toList

- Een niet wijzigbare lijst maken

8 Hoofdstuk 8: String en Reguliere Expressies

8.1 Reference equality

```
String s1 = "Hallo";
String s2 = "Hallo";
System.out.print(s1 == s2)
```

- Hier zal dit werken, hier krijg je `true`

```
String s1 = new String("Hallo");
String s2 = new String("Hallo");
System.out.print(s1 == s2)
```

- Hier zal dit niet werken, hier krijg je `false`
- Dit kan je oplossen door `System.out.print(s1.equals(s2))` te herschrijven

8.2 Immutiale (tip: *finale*)

- Strings zijn **immutible**, een String kan je nooit **wijzigen**
- Er zijn wel methodes die een nieuwe String maken van een al-bestaande String

```
s1 = s1.replace("a", "e");
```

8.3 Stringbuilder

- Hier vindt je gelijkaardige methodes als in String, maar ook nieuwe
- bv. **append** en **reverse**, hierdoor kan je elk soort datatype toevoegen aan een String zonder dat het problemen geeft.

```
Stringbuilder s1 = new StringBuilder("Hallo");
s1.append(1).append(true).append(3.4).append("abc");
```

8.4 Reguliere expressies

- Dit gebruik je of een String voldoet aan een bepaald aantal voorwaarden

```
String regulierexpressie = "abc";
String mijnTekst = "abc";

// hier schrijven we of onze String voldoen aan de volgende voorwaarden:
// veranderen we hier iets aan de waarde, krijgen we false
System.out.println(mijnTekst.matches(regulierexpressie));
```

8.5 Characterclass - lijstje van karakters die er zeker in moet inzitten als xde letter - dit is hoofdlettergevoelig

```
String regulierexpressie = "a[xyz]c";
String regulierexpressie = "a[^xyz]c";
String mijnTekst = "abc";
```

8.6 Characterclass - een bereik van letters en ook het omgekeerde

```
String regulierexpressie = "a[a-z]c"; // moet bevatten
String regulierexpressie = "a[^a-z]c" // niet bevatten
String regulierexpressie = "a[^a-zA-Z]c"; // niet bevatten inclusief
   hoofdletters
String regulierexpressie = "a[.]c"; // gelijk welk karakter
String regulierexpressie = "a[\D]c"; // allesbehalve een cijfer
String regulierexpressie = "a[*]c"; // 0 of meerdere karakters
String regulierexpressie = "a[?]c"; // het mag of er mag niet staan
String regulierexpressie = "a[a\\d]+c"; // 1 of meerdere keren moet dit
   groepje voorkomen
String mijnTekst = "abc";
```

8.7 Characterclass - Klasse Z

- Dit lijkt op dat van Linux, omdat dit soort expressies komt algemeen voor in andere talen (ook buiten codetalen)

```
String regulierexpressie = "a[X?]c"; // het mag of er mag niet staan
String regulierexpressie = "a[X*]c"; // komt geen keer voor of niet
String regulierexpressie = "a[X+]c"; // moet 1 of meerdere keren voorkomen
   of niet
String regulierexpressie = "a[X{n}]c"; // exact een getal een aantal keer
String regulierexpressie = "a[X{n, }]c"; // minstens een bepaald getal
String regulierexpressie = "a[X{n, m}]c"; // van een getal tot een getal

String mijnTekst = "abc";
```

8.8 Characterclass - Voorbeelden

- Dit lijkt op globbing van Linux, omdat dit soort expressies komt algemeen voor in andere talen (ook buiten codetalen)

```
String regulierexpressie = "[A-Z].*"; // moet beginnen met een hoofdletter
   gevolgd door een random aantal karakters
String regulierexpressie = "[A-Z].{5,7}"; // moet beginnen met een
   hoofdletter en is 6 letters lang door een random aantal karakters
String regulierexpressie = "([A-Z].{5,7} ){2}"; // moet beginnen met een
   hoofdletter en is 6 letters lang door een random aantal karakters
   gevolgd door een spatie en dat verwacht je 2x, maar einde is ook een
   spatie
String regulierexpressie = "(ab)+\\d"; // a of b of ab 1 of meerdere keren
   gevolgd door een cijfer
String regulierexpressie = "a(b|c)d"; // starten met a, b of c en eindigen
   op d

String mijnTekst = "abc";
```

8.9 Pattern en Matcher

- Voor één grote string kun je een Pattern maken
- Hier vraag je of de tekst overeenkomt met de reguliere expressie

```
Pattern pattern = Pattern.compile(regulierexpressie);
Matcher matcher = pattern.matcher(mijnTekst);
```

```
System.out.println(matcher.find());
```

- Hier vind je alle stukjes van je String die wel voldoen aan de reguliere expressie

```
while (matcher.find)
    System.out.printf("%s gevonden die volgdoet aan de regulierexpressie",
       matcher.group())
```

8.10 Oefeningen

```
    // nemen middelste karakter
    return inhoud.charAt(inhoud.length() / 2);

    // inhoud omdraaien
    return new StringBuilder(inhoud).reverse().toString();

    // palindroom
    return inhoud.equals(geefOmgekeerdeInhoud());

    // karakters vervangen
    return inhoud.replace(Character.toLowerCase(letter1), Character.toLowerCase
       (letter2));

    // een woord splitten - hier kunnen ook reguliere expressies inzitten
    return inhoud.split(woord);
```

8.11 Oefening - een zin zijn aantal letters counten

```
    public String geefLetterRapport() {
        StringBuilder sb = new StringBuilder("Aantal klinkers: ");

        int aantalKlinkers = 0;
        int aantalMedeklinkers = 0;
        int aantalKleineletters = 0;
        int aantalHoofdletters = 0;
        int aantalCijfers = 0;
        int aantalSpecialeChars = 0;
        int aantalWoorden = 1;

        for (String letter : inhoud.split("")) {
            if (letter.matches("[aeuioAEUIO]"))
                aantalKlinkers++;
            if (letter.matches("[a-zA-Z&&[^aeiouAEIOU]]"))
                aantalMedeklinkers++;
            if (letter.matches("[a-z]"))
                aantalKleineletters++;
            if (letter.matches("[A-Z]"))
                aantalHoofdletters++;
            if (letter.matches("\\d"))
                aantalCijfers++;
            if (letter.matches("[^a-zA-Z0-9]"))
                aantalSpecialeChars++;
            if (letter.matches(" "))
                aantalWoorden++;
        }
```

```
        sb.append(aantalKlinkers).append("\n").append(String.format("Aantal
           medeklinkers: %d", aantalMedeklinkers))
                .append(String.format("%nAantal kleine letters: %d",
                   aantalKleineletters))
                .append(String.format("%nAantal hoofdletters %d",
                   aantalHoofdletters))
                .append(String.format("%nAantal cijfers: %d", aantalCijfers))
                .append(String.format("%nAantal speciale chars: %d",
                   aantalSpecialeChars))
                .append(String.format("%nAantal woorden: %d", aantalWoorden));

        return sb.toString();
    }
```

9 Hoofdstuk 9: Bestanden

9.1 URL naar het bestand:

```
// src/packagename/naamBestand
private final static String PATHNAME = "src" + File.separator + "tekst" +
    File.separator + "clients.txt";
```

9.2 OutputStream en InputStream

- Outputstream (wordt gebruikt door een formatter) + (ObjectInputStream)
  - Zet tekst vanuit een bestand naar Java
- Inputstream (wordt gebruikt door een Scanner) + (ObjectOutputStream)
  - Zet tekst vanuit Java naar een bestand

9.3 Serializable

- Bestanden die tekst halen uit een bestand moeten gebruik maken van Interface `implements Serializable`

- Om bestand te lezen en halen uit een bestand schrijf je:
  - `readObject`
  - `writeObject`

9.4 Een functie maken dat een binair bestand aanmaakt

```
public void serialiseerObjectPerObject(Collection<Speler> spelerslijst,
    String naamBestand) {
     String URL1 = "src" + File.separator + "bestanden" + File.separator +
         naamBestand;
     try (ObjectOutputStream oos = new ObjectOutputStream(Files.
         newOutputStream(Path.of(URL1)))) {
          for (Speler s : spelerslijst) {
              oos.writeObject(s);
          }
     } catch (IOException e) {
          System.err.println(e);
     }
}
```