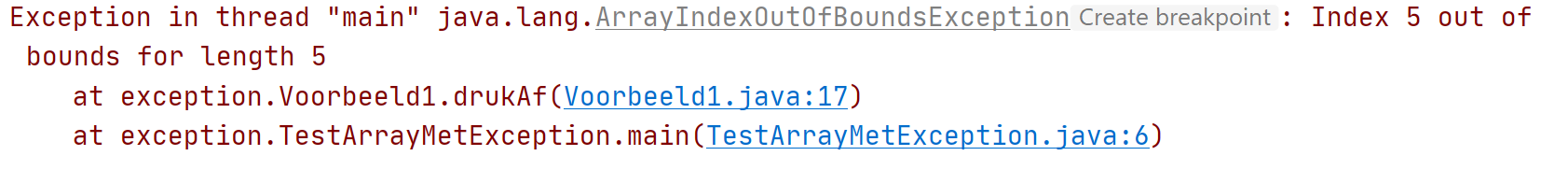
**Opvangen van een exception**

**Voorbeeld 1**  
Deze exception bespraken we reeds tijdens de laatste les programming fundamentals.

We bekijken onderstaand voorbeeld en maken bewust een fout:

public class Voorbeeld1 {  
 private int[] lijstGetallen = new int[5];  
  
 public Voorbeeld1() {  
 for (int i = 0; i < 5; i++) {  
 lijstGetallen[i] =  
 Integer.*parseInt*(JOptionPane.*showInputDialog*("Geef getal"));  
 }  
 }  
  
 public void drukAf() {  
 for (int i = 0; i <= 5; i++) {  
 System.*out*.println(lijstGetallen[i]);  
 }  
 }  
  
}

**Output:**



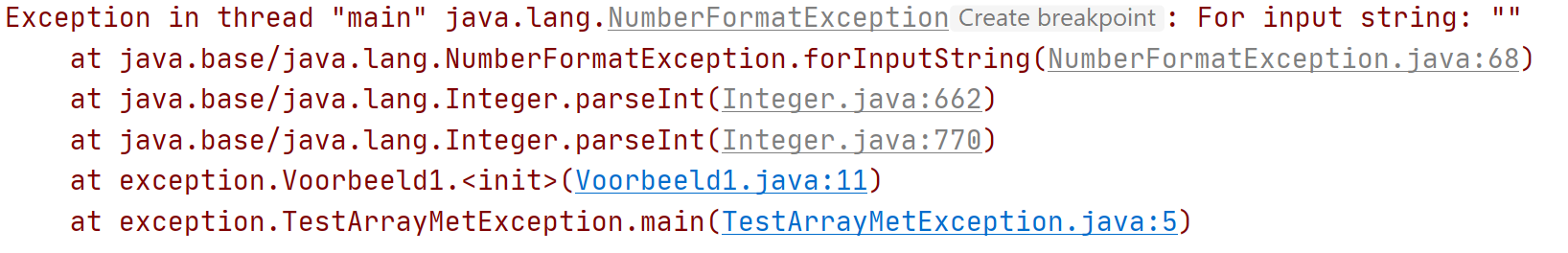
**Uitleg:**

De teller i loopt t.e.m. 5, waardoor de laatste sout lijstGetallen[5] probeert af te drukken. Dit kan niet want 4 is de laatste plaats in de array waar naar verwezen kan worden. De array bevat in dit geval op die plaats het getal 12 = lijstGetallen[4].

Gevolg is dat er een exception gegenereerd wordt. Een exception is een object dat informatie bevat over de fout die zich heeft voorgedaan. Dit exception-object kunnen we opvangen en netjes afhandelen.   
Aangezien het niet de bedoeling is om dergelijke fouten te maken, zullen we bovenstaande exception niet opvangen.

**Voorbeeld 2**

We gaan verder met bovenstaand voorbeeld.  
Wanneer in het input-dialoog-venster geen waarde of een niet-numerieke waarde wordt ingegeven, krijgen we de volgende exception.



We vangen deze exception op in Voorbeeld2 en we tonen een gepaste foutmelding met JOptionPane.  
Denk er aan dat je de volledige array moet invullen en je dus 5 juiste getallen zal moeten opgeven.

package exception;  
  
import javax.swing.\*;  
  
public class Voorbeeld2 {  
 private int[] lijstGetallen = new int[5];  
  
 public Voorbeeld2() {  
 for (int i = 0; i < 5; i++) {  
 try {  
  
 lijstGetallen[i] =  
 Integer.*parseInt*(JOptionPane.*showInputDialog*("Geef getal"));  
 } catch (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(null, "je moet een getal ingeven");  
 i--;  
 }  
 }  
 }  
  
 public void drukAf() {  
 for (int i = 0; i <= 4; i++) {  
 System.*out*.println(lijstGetallen[i]);  
 }  
 }  
}

**Uitleg code:**

try-blok: dit wordt altijd uitgevoerd. Binnen deze code kunnen exceptions zich voordoen.   
catch-block: dit wordt uitgevoerd wanneer er een exception optreedt. Het try-block stopt dan, de instructies die horen bij het catch block worden uitgevoerd.  
Treedt er geen exception op in het try-blok, dan wordt het catch blok niet uitgevoerd.  
Indien er na het catch block nog instructies zijn, worden deze na de uitvoering van het catch block uitgevoerd.

Conclusie: een statement waarin een exception zou kunnen optreden en die je niet kan vermijden zoals hier, plaats je in een try-blok. Het opvangen van de exception doe je in een catch-blok.

**Musicorganiser.**

**Deze code komt uit het boek Programmeren in Java met BlueJ, van David J. Barnes en Michael Kolling.**

We leerden vorige week de documentatie van Java te raadplegen en klassenbibliotheken te gebruiken.  
Jullie zouden nu in staat moeten zijn om bestaande code te interpreteren en te begrijpen en uit te breiden en aan te passen. Het is niet erg indien je niet alles begrijpt.   
Probeer te antwoorden op onderstaande vragen om een beter inzicht te krijgen hoe de code in elkaar zit.

**Vraag 1**  
Maak een UML schema van de klassen van MusicOrganiser.   
 **Bespreking class TrackReader.**

**Vraag 2**public ArrayList<Track> readTracks(String folder, final String suffix)

Wat betekent final?  
  
 **Vraag 3**File audioFolder = new File(folder);  
  
Zoek deze constructor op. Let op, er zijn verschillende constructoren, de welke is hier gebruikt?

**Vraag 4**

FilenameFilter fn = new FilenameFilter() {  
 @Override  
 public boolean accept(File dir, String name) {  
 return name.toLowerCase().endsWith(suffix);  
 }  
};

Zoek de interface FilenameFilter. Wat is een interface? Wat is het verschil tussen een interface en een class? Zoek op.   
<https://www.w3schools.com/java/java_interface.asp>

Welke methods bevat deze interface? Meer dan 1? Wat doet bovenstaande interface?

**Vraag 5**

FilenameFilter fn = new FilenameFilter() {  
 @Override  
 public boolean accept(File dir, String name) {  
 return name.toLowerCase().endsWith(suffix);  
 }  
};

Wat betekent @Override?   
Is het een goed idee om deze annotatie te laten staan in de code?  
Denk aan de String toString-functie.  
Plak onderstaande code in de class Track  
Waarom levert volgende code een fout op?   
@Override wordt rood onderlijnd.  
  
   
@Override  
public String ToString() {  
 return "Track{" +  
 "artist='" + artist + '\'' +  
 ", title='" + title + '\'' +  
 ", filename='" + filename + '\'' +  
 '}';  
}

**VRAAG 6**

private Track decodeDetails(File file) {  
 *// The information needed.* String artist = "unknown";  
 String title = "unknown";  
 String filename = file.getPath();  
  
 *// Look for artist and title in the name of the file.* String details = file.getName();  
 String[] parts = details.split("-");  
  
 if (parts.length == 2) {  
 artist = parts[0];  
 String titlePart = parts[1];  
 *// Remove a file-type suffix.* parts = titlePart.split(".");  
 if (parts.length >= 1) {  
 title = parts[0];  
 } else {  
 title = titlePart;  
 }  
 }  
 return new Track(artist, title, filename);  
}

Wat doet de split-functie? Zoek op.  
Hoe verloopt de code met het volgende argument?  
“BigBillBroonzy-BabyPleaseDontGo1.mp3”

Waarom staat deze functie private? Wat is het verschil met public?  
Indien je deze functie rechtstreeks wil aanroepen vanuit de Test-klasse, maak ze dan private en print het object dat terug gegeven wordt uit.

**En nu gaan we zelf programmeren en de bestaande code uitbreiden.**

**Oef1**Voeg een mp3-bestand toe en zorg ervoor dat de filenaam juist is.  
  
**Oef2**Wijzig in de class MusicOrganiser de functie allSamples zodat de volledige lijst wordt afgespeeld.

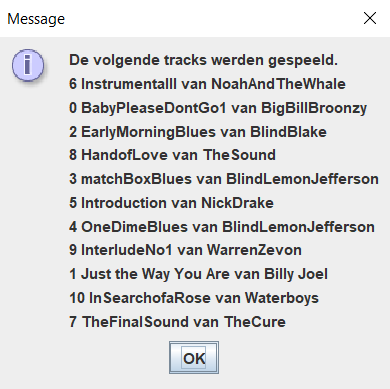
**Oef3**

Maak een functie allSamplesShuffle die hetzelfde doet als vorige maar steeds in een andere en dus willekeurige volgorde.

**Oef4**Maak een functie playArtist die alle samples van een bepaalde artiest afspeelt. Zorg ervoor dat de functie niet hoofdlettergevoelig is en geen rekening houdt met leestekens en blanco’s.   
Testen!

**Oef5**

Verander de functie allSamplesShuffle() zodat deze functie vóór het afspelen een overzicht toont zoals hieronder van de gespeelde tracks met hun bijhorende volgnummer in de ArrayList tracks.



**Oef6**Wat moet er veranderen indien je verschillende gebruikers hebt, die allemaal hun eigen Musicorganiser hebben?   
Van elke gebruiker wil ik zijn naam kunnen opvragen en zijn afspeellijst. Return dit in de vorm van een String.  
Ter vereenvoudiging om te testen, mag je elke gebruiker dezelfde afspeellijst geven. Normaal heeft elke gebruiker zijn eigen afspeellijst, maar dan moet je te veel mp3-bestanden klaarzetten.

**Oef7**

Wat verandert er wanneer ook het jaartal wordt opgegeven. Maw, per track willen we bijhouden :   
artiest, titel, jaartal, filename.  
De naam van het mp3 bestand is dan als volgt : naam-titel-jaartal.mp3  
Test uit.

**Oef8**

public void listTrack(int index) {  
 if (indexValid(index)) {  
 System.*out*.print("Track " + index + ": ");  
 Track track = tracks.get(index);  
 System.*out*.println(track.getDetails());  
 }  
}

Verander bovenstaande functie zodat je geen parameter opgeeft, maar met JOptionPane een nummer opvraagt.  
Toon bijvoorbeeld de volgende boodschap : “geef een nummer op tussen 0 en 12”.   
Vang met een exception een verkeerd formaat van de input op. (lege input of geen numerieke input)

Blijf een getal vragen totdat het een juist getal is.

**SWITCH CASE**

**Syntax**

**switch(*expression*) {**

**case x:**

***// code block***

**break;**

**case y:**

***// code block***

**break;**

**default:**

***// code block***

**}**

* De switch expressie wordt geëvalueerd.
* De waarde van de expressie wordt vergelijken met de waarde van elke case.
* Indien de waarden overeenkomen, dan wordt het overeenkomstige blok uitgevoerd.
* Het break-statement verlaat de volledige switch structuur. Er worden maw, geen andere cases meer gecheckt.
* Het code block bij default wordt uitgevoerd indien er geen enkele match met een case is.

Switch case wordt vaak gebruikt ipv een ingewikkelde if-structuur.  
Switch wordt door de meeste developers beschouwd als bad practice, je moet natuurlijk wel weten wat het is. Meestal zijn er betere oplossingen voor switch-case. Dit zie je in latere cursussen.

public class SwitchVoorbeeld {  
 public static void main(String[] args) {  
 int leeftijd = Integer.*parseInt*(JOptionPane.*showInputDialog*("Wat is je leeftijd"));  
 switch (leeftijd)  
 {  
 case(75):  
 System.*out*.println("vanaf nu enkel nog genieten");  
 break;  
 case(50):  
 System.*out*.println("nu begint het pas!");  
 break;  
 case(18):  
 System.*out*.println("eindelijk volwassen");  
 break;  
 default:  
 System.*out*.println("elke leeftijd heeft zijn charme");  
 }  
 }  
}

**Rekenspel**

Werk object georiënteerd en denk na welke klassen en private variabelen nodig zijn.  
Let op het correct gebruik van lokale variabelen (maak dus niet voor elke var. die je nodig hebt een private variabele aan.

Met al dat afstandsonderwijs is er nood aan gemakkelijke applicaties om de tafels van vermenigvuldiging te oefenen. Ook optellen, aftrekken en delen moet flink ingeoefend worden.   
Wegschrijven van de gegevens naar een database leren we in kwartaal 3. Je houdt alles tijdens de run van het programma bij in ArrayLists en Arrays.

Werk met exceptions om de invoer te checken.  
 **Stap 1**

Maak een klasse Leerling.  
  
Deze klasse heeft als private variabele de naam van de leerling.  
Zoek zelf uit welke andere private variabelen nog nodig zijn. Het kan zijn dat bijkomende klassen nodig zijn.  
Alle behaalde scores moeten bijgehouden worden. Ook moet bij elke score worden bijgehouden welke bewerking werd geoefend.   
 De leerling moet een keuze kunnen maken tussen \*, /, + en -.  
De resultaten van de sommen zijn maximaal 100.   
Bij aftrekken mogen de resultaten niet negatief zijn.  
Bij delen moeten de getallen deelbaar zijn door elkaar.  
Bij vermenigvuldigen worden alle tafels tot en met 10 ondervraagd.  
  
De leerling krijgt 10 sommen en krijgt telkens 2 keer de kans om te antwoorden. Indien de eerste keer juist, dan krijgt hij 2 punten.  
Indien de som na de tweede keer juist is, dan krijgt hij 1 punt. Indien de som niet juist is, dan krijgt hij geen punten en wordt de juiste uitkomst getoond.  
Na 10 sommen krijgt de leerling de eindscore op 20.   
Indien hij 20/20 behaalt, wordt dit beloond met een muziekje.  
Voorzie ook een aantal bemoedigende zinnen die je random toont, zodat de kinderen niet steeds dezelfde boodschap krijgen.  
  
Voorzie drie soorten zinnen (verschillende per keer).  
Voor scores tussen 16 en 20.  
Voor scores tussen 10 en 15.  
Voor scores onder de 10.   
Later lossen we dit op aan de hand van een HashMap.  
Nu kan je dat oplossen met arrays. Gebruik hiervoor verplicht een array.

**Stap 2**

Vraag aan de leerling of hij opnieuw een reeks sommen wil maken.   
Hij kan antwoorden met Yes, Ja, JA, j, ja, Nee, Neen, N, no, No. Wees hier soepel.

Geef op het einde een overzicht van de behaalde scores per keer dat hij het rekenspel gespeeld heeft, tevens over welke bewerking het ging.

**Stap 3**

De leerling moet nu ook de mogelijkheid krijgen om alle bewerkingen door elkaar te krijgen.  
 **Stap 4**Zorg dat je deze overzichten kan bijhouden per klas.

TIP : je hebt dus alvast de klasse Leerling en de klasse Klas nodig.  
Het volgende moet mogelijk zijn.  
Maak hiervoor de nodige functies.  
-De leerkracht kan op basis van de naam van de leerling een overzicht opvragen van de behaalde scores. Op dat moment kan de leerkracht zien wat de problemen zijn en of de leerling voldoende geoefend heeft.  
-Toon per leerling de gemiddelde scores.  
-Toon alle leerlingen die minder dan het gemiddelde geoefend hebben.