SuperCheatSheet Java Advanced  
   
Basic Shit  
  
**begin:**  
public static void main(String[] args)  
{  
}  
  
**class:**  
public class Person  
{  
}  
  
**constructor:**  
public Person(String name , int age) constructor  
{  
 super(name); van superconstructor  
 this.age = age; van eigen constr  
}  
  
**getter:**  
public int getAge() { return age;}  
  
System.out.println(p1); uitprinten  
  
**Interface :**   
- lege klasse met lege methods  
- ‘implements’  
  
Een klasse met een andere uitbreiden:  
‘extends’  
  
**lus:**  
for (int i = 1 ; i < values.size() ; i ++)  
{  
}

**toString overschrijven binnen een klasse:**  
public String toString()  
{  
 return “Person{“ + “name =’ “ + getName() + ‘\’ ‘ + “, age=” + getAge() + ‘}’;  
}  
eventueel met “\n” en “\t”  
  
**array:**  
Person[] persons = new Person[2];  
persons[0] = don;  
persons[1] = bert;  
  
**random**  
int randomWithRange(int min, int max)   
{   
 int range = (max - min) + 1;   
 return (int)(Math.random() \* range) + min;   
}  
of  
Random rnd = new Random();

rnd.next(getal) // getal zit er zelf niet bij dus als getal 100 is 0-99

of random van letters

Random r = new Random();  
char c = (char)(r.nextInt(26) + ‘a’);  
  
  
**testen of iets leeg is:**  
while ( teller != null )  
{  
}  
  
Collections  
  
List  
  
import java.util.List;  
  
List <Person> lijst = new ArrayList<Person>(); Aanmaken  
lijst.add(p1); Toevoegen aan de lijst  
lijst.get(0); Geeft eerste element weer  
lijst.isEmpty(); checken of iets in de lijst zit  
  
  
bv.  
for ( Person person : lijst)  
{  
 System.out.println(person);  
}  
  
Set  
--🡪 geen dubbels, geen volgorde  
  
import java.util.Set;  
  
Set <Person> lijstje = new HashSet <Person>(); Aanmaken  
lijstje.contains(donDraper); True/false of het in de set zit  
  
Map  
--🡪 unieke key mappen aan een value  
  
import java.util.Map;  
  
Map<String, Person> lijstje = new HashMap<String, Person>(); Aanmaken  
  
lijstje.put(donDraper.getName(), donDraper); Toevoegen   
!! indien opnieuw met zelfde key maar andere value wordt key behouden maar value overschreven!!  
lijstje.get(“donDraper”); zoekt Person op van key  
  
for (String name : lijstje.keySet())  
{  
system.out.println(name);  
}  
  
for (Person person : lijstje.values())  
{  
system.out.println(person);  
}  
  
for (Map.Entry<String, Person> entry : lijstje.entrySet())  
{  
system.out.println(entry);  
system.out.println(getKey());  
system.out.println(getValue());  
}  
  
Generics  
  
Comparator  
- 1 kleiner dan, 0 gelijk en 1 groter dan  
  
import java.util.Comparator;  
  
Collections.sort(lijstje,new AgeComperator());  
  
public class AgeComperator implements Comparator<Person>  
{  
 public int comparator(final Person left, final Person right)  
 {  
 return Integer.compare(left.getAge(),right.getAge());  
 }  
}  
  
**of omgekeerd sorteren:**  
Collections.sort(lijstje,new ReverseComparator<>(new AgeComperator()));  
  
public class ReverseComperator<T> implements Comparator<T>  
{  
 private final Comparator<T> delegateComparator;  
  
 public ReverseComparator(final Comparator<T> delegateComparator)  
 {  
 this.delegateComparator = delegateComparator;  
 }  
  
 public int comparator(final T left, final T right)  
 {  
 return -1 \* delegateComparator.compare(left,right);  
 }  
}  
  
Type Bound  
  
CompareTo  
  
public class SortedPair**<T extends Comparable<T> >**  
{  
 private final T first;  
 private final T second;  
  
 public SortedPair (T left, T right)  
 {  
 if (left.compareTo(right) < 0 )  
 {  
 first = left;  
 second = right;  
 }  
 else  
 {  
 first = right;  
 second = left;  
 }  
 }  
  
 public T getSecond() { return second; }  
 public T getFirst() { return first; }  
}  
  
Minimum  
  
final Person youngest = min(lijstje , new AgeComperator());  
  
public static **<T>** **T** min(List **<T>**values , Comparator**<T>** comparator)  
{  
 if ( values.isEmpty())  
 {  
 throw new IllegalArgumentException(“List is empty”);  
 }  
 **T** lowestElement = values.get(0);  
 for (int I = 1 ; I < values.size() ; i++)  
 {  
 final **T** element = values.get(i);  
 if ( comparator.compare(element, lowestElement) < 0)  
 {  
 lowestElement = element;  
 }  
 }  
 return lowestElement;  
 }  
  
**extra**  
List<Integer> numbers = new ArrayList<>();  
numbers.add(1);  
numbers.add(2);

Numbers.add(3);  
System.out.println(min(numbers, Integer::compare));  
  
WildCards  
  
List <? extends Person> subklassen van Person  
 om iets uit de parameter te halen  
List <? super Person> Person en Parent klassen ervan  
 om iets in de parameter te steken  
List <?> = ? extends Object  
Person met subklassen Employee, Partner,…  
  
List<Person>  
wordt  
List< **? extends** Person>  
wanneer ook Employee en Partner hier gebruikt kunnen worden  
  
**!! enkel gebruiken wanneer later geen referentie nodig is.**

**In dat geval T gebruiken zodat men kan refereren !!**  
  
Threads   
  
**Extra:**  
- Elke thread heeft een uniek thread id.  
Dit kan men krijgen door **Thread.currectThread().getId()** .  
- Men kan een thread een naam geven door **.setName()** en een naam opvragen kan men door **Thread.currentThread().getName().**  
- toestanden **NEW, RUNNABLE (READY of RUNNING) en TEMRINATED**.  
- **.isAlive()** geeft true wanneer de thread RUNNABLE is.  
- .**getState()** geeft de toestand terug Thread.State.NEW ,Thread.State.RUNNABLE, Thread.State.BLOCKED, Thread.State.WAITING, Thread.State.TIMED\_WAITING , Thread.State.TERMINATED.  
- .**setPriority()** verandert de prioriteit van de thread (min 1, norm 5 , max 10) variable Thread.MIN\_PRIORITY  
- **Thread.yield()** laat een thread uit de nunning state gaan zodat een andere thread met gelijke prioriteit kan draaien in zijn plaats ipv te wachten   
- .**setDaemon(true)** maakt van een gewone een Daemon thread. De applicatie wacht niet dat de thread klaar is en sluit wanneer alle niet-daemon threads klaar zijn.  
- **System.exit()** forceert het sluiten van de applicatie onafhankelijk van wat er nog loopt  
- **Thread.sleep()** laat een thread uit Runnable in slaap toestand gaan. Dit gaat steeds over de actieve tohread. De nieuwe toestand is dan **TIMED\_WAITING**. Hier kan met een tijd in meegeven in miliseconden. Men kan ze ook vroegtijdig wakkermaken door **.interrupt()**. Dan komt deze terug op READY. Wanneer .interrupt() wordt gebruikt genereert dit ook een **InterruptException**.   
**!! .sleep() altijd met try/catch voor InterruptException !!**  
- .**join()** laat een volgende thread wachten tot de huidige thread op TERMINTED komt. De status van deze thread is dan **WAITING** . join() kan enkel op een RUNNING thread en na de .join() komt de wachtende thread terug op READY te staan net als met .interrupt(). Men kan hier ook een maximale tijd meegeven in milisec. Indien dit gebeurt dan is de status ook **TIMED\_WAITING** zoals bij .sleep(). Ook hier kan men **.interrupt()** gebruiken om de thread uit WAITING te halen.  
**!! .join() altijd met try/catch voor InterruptException !!**  
  
**gebruik**  
  
public class ContinuousPrinter **extends Thread** {  
 @Override  
 public void run() {  
 for ( int i = 0 ; I < 100 ; i++) { acties van de thread  
 System.out.println(“Hello”)  
 }  
 super.run();  
 }  
}  
  
public class MainClass {  
 public static void main(String[] args) {  
 ContinuousPrinter cp1 = new ContiniuousPrinter();  
  
 ContinuousPrinter cp2 = new ContiniuousPrinter();  
 cp1**.start();** cp2**.start();**  
. }  
}  
  
**of beter**  
  
public class ContinuousPrinter **implements Runnable** {  
 @Override  
 public void run() {  
 for ( int i = 0 ; I < 100 ; i++) { acties van de thread  
 System.out.println(“Hello”)  
 }  
 }  
}  
  
public class MainClass {  
 public static void main(String[] args) {  
 ContinuousPrinter cp1 = new ContiniuousPrinter();  
 **Thread** thread1 **= new Thread(**cp1**);**  
 ContinuousPrinter cp2 = new ContiniuousPrinter();  
 Thread thread2 = new Thread(cp2);  
 thread1**.start();** thread2**.start();**  
. }  
}  
  
**met Executor**  
  
import java.util.concurrent.ExecutorService;  
  
public class ContinuousPrinter **implements Runnable** {  
 @Override  
 public void run() {  
 for ( int i = 0 ; I < 100 ; i++) { acties van de thread  
 System.out.println(“Hello”)  
 }  
 }  
}  
  
public class MainClass {  
 public static void main(String[] args) {  
 ContinuousPrinter cp1 = new ContiniuousPrinter();  
 **ExecutorService executor = Executors.newFixedThreadPool(3);** //hier dus max 3 tegelijk  
 **executor.submit(**cp1**);**  
 executor.submit(cp1);  
 executor.submit(cp1);  
 executor.submit(cp1);  
 executor.submit(cp1);  
 executor.submit(cp1);  
 executor.submit(cp1);  
 **executor.shutdown();** vanaf hier kan niks toegevoegd worden  
 }  
}  
  
**extra:**  
ipv pool ook **Executor.newSingleThreadExecutor();** mogelijk.  
Hierbij gaat hij de threads 1 per 1 eerst afwerken terwijl de main verder loopt.  
  
Threads – Synchronization (several threads using same data at same time)  
  
public **synchronized** void UseMethod() {  
 \*\*actions in method\*\*  
}  
  
Concurrent collections (collections met synchronized)  
  
List<Integer> list = new ArrayList<>();

List<Integer> slist = Collections.synchronizedList(list);  
  
andere:  
synchronizedCollection()  
synchronizedList()  
synchronizedNavigableMap()  
synchronizedNavigableSet()  
synchronizedSet()  
synchronizedSortedMap()  
synchronizedSortedSet()  
  
import java.util.concurrent.\*;  
  
ConcurrentHashMap()  
ConcurrentSkipListMap()  
ConcurrentSkipListSet()  
CopyonWriteArrayList()  
CopyonWriteArraySet()  
(p170)  
  
List<Integer> list = new CopyOnWriteArrayList();  
  
  
Lambda (nog verder afwerken)  
  
List <Person> people = …. ;  
people.forEach(System.out::println); print elk element uit people met toString  
people.removeIf(person -> person.getAge()<20); Alle jonger dan 20 uit de lijst halen  
  
Lambda & Thread combineren (p30-35)  
  
CharacterPrinter printer1 = new CharacterPrinter(‘\*’,100);  
Thread thread1 = new Thread( printer1);  
thread1.start();  
  
Kan men ook schrijven als  
  
Thread thread1 = new Thread( () -> print(‘\*’,100) );  
thread1.start();  
  
Method Reference (36-41)  
  
**objectName::methodName**  
s -> padder.pad(s)  
wordt   
padder::pad  
  
**ParameterType::methodName**  
s -> s.toUpperCase()  
wordt  
String::toUpperCase  
  
**ClassName::new**  
s -> new Long(s)  
wordt  
Long::new  
  
  
IO Streams  
  
import.java.io.IOException;  
  
InputStream for bytes (raw data)  
OutpuStream  
  
Reader text based streams for ascii  
Writer  
  
**lezen:**List<Person>klantenLijst = getLijstFromFile();  
private String getLijstFromFile()  
{  
 List <T> invoerLijst = new ArrayList<T>();  
   
 try (FileReader reader = new FileReader(“C:\\temp\\test.txt”);  
 BufferedReader bReader = new BufferedReader(reader); )  
 {  
 String ingelezenLijn = bReader.readLine();  
 while ( ingelezenLijn != null )  
 {  
 String[]words = ingelezenLijn.split(“;”);  
 T p = new T (words[0], words[1], Integer.parsInt(words[2]));  
 invoerLijst.add(p);  
 ingelezenLijn = bReader.readLine();  
 }  
 return invoerLijst;

} catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return “”;  
}  
  
**wegschrijven:**  
private void wegSchrijven(List<T> Lijst)  
{  
 try ( PrintWriter writer = new PrintWriter(FileWriter(“C:\\temp\\outputpage.txt”)); )  
 {   
 String lijn = “”;  
  
 for ( T p : Lijst)   
 {  
 lijn = String.format(“%s;%s;%d”), p.getVeld() , p.getTweedeVeld() , p.getGetal() ) ;  
 writer.write(lijn);  
 writer.newLine();  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}  
  
**voorbeeld lezen met extra reader en naar hoofdletters zetten:**  
  
List<Person> klantenLijst = getLijstFromFile();  
  
public class CapitalizationReader extends BufferedReader {  
 @Override  
 public String readLine() throws IOException{  
 String line = super.readLine();  
 if (line != null)  
 return line.toUpperCase();  
 return null;  
  
 }  
 public CapitalizationReader(Reader in)  
 {  
 super(in);  
   
 }  
}  
  
private String getTextFromFile()  
{  
 List<T> invoerLijst = new ArrayList<T>();  
 try ( FileReader reader = new FileReader(“C:\\temp\\test.txt”);  
 BufferedReader bReader = new BufferedReader(reader);  
 CapitalizationReader capReader = new CapitalizationReader(bReader) ;)  
 {  
  
   
 String ingelezenLijn = capReader.readLine();  
 while (ingelezenLijn != null )  
 {  
 String[]words = ingelezenLijn.split(“;”);  
 T p = new T (words[0],words[1],Integer.parsInt(words[2]));  
 invoerLijst.add(p);  
 ingelezenLijn = capReader.readLine();  
 }  
 return invoerLijst;

} catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return “”;  
}  
  
File IO Binair - Serialization  
  
**wegschrijven met serilization**  
import java.io\*;  
import java.time.\*;  
  
public class WriteObjectApp (  
 public static void main(String[] args) {  
 String tekst = new String(“this is some tekst”);  
 LocalDateTime date = LocalDateTime.now;  
   
 try ( FileOutputStream file = new FileOutputStream(“MyFile.ser”);  
 ObjectOutputStream out = new ObjectOutputStream(file);)  
 {  
 out.writeObject(text);  
 out.writeObject(date);  
 }  
 catch (IOException ex)   
 {  
 System.out.println(ex.getMessage());  
 }  
 }  
}  
  
**lezen met serilization**  
import java.io\*;  
import java.time.\*;  
  
public class ReadObjectApp (  
 public static void main(String[] args) {  
   
 try ( FileInputStream file = new FileInputStream(“MyFile.ser”);  
 ObjectInputStream in = new ObjectInputStream(file);)  
 {  
 String tekst = (String) in.readObject();  
 LocalDateTime date = (LocalDateTime) in.readObject;  
 }  
 catch (IOException ex)   
 {  
 System.out.println(ex.getMessage());  
 }  
 }  
}  
  
**extra info**  
om een object klasse serialiseerbaar te maken gewoon  
 **implements Serializable**  
bij de hoogste klasse ervan zetten. Alle subklassen erven dit door.

## Streams (p 69)

**Stream aanmaken**

List<String> words = new ArrayList<>();  
Stream<String> stream = words.Stream();

**Stream afdrukken**

words.stream().forEach(System.Out::println);

**Filters Toepassen en direct afdrukken**

words.stream().Filter(s -> s.contains("e")).forEach(System.Out::println);  
words.stream().Filter(s -> s.length() > 3)).forEach(System.Out::println);

**Streams - omzetten van een Object collection naar een collection van een ander datatype**

words.stream().mapToInt(String::length)   
words.stream().mapToDouble   
words.stream().map(String::toUpperCase)

**Streams collecteren (p 71 -72)**

List<String> shortWords = words.stream().filter(s->s.length < 4) .collect(Collectors.ToList());  
Nieuwe List met woorden kleiner dan 4 lengte.  
  
**Streams combineren**  
words.stream().filter( s -> s.length() > 3 ) stap 1  
 .filter( s -> s.contains(“e”) ) stap 2  
 .forEach(System.out::println); stap 3  
**Stream – reduceren van van object collection naar int collection naar int varable (p70)**

Int length = words.stream().mapToInt(String::length**.sum();**  
  
String total = words.stream().reduce(“”, (String t, String u ) -> t+u);  
waarbij “” de startwaarde is  
 t tussenliggende waarde  
 u te behandelen object  
  
**Parallel  
Indien men onderdelen van de stream parallel wil laten behandelen ipv sequentieel**  
  
words.**parallelStream().**filter( s -> s.length() > 3 ) stap 1  
 .filter( s -> s.contains(“e”) ) stap 2  
 .forEach(System.out::println);  
  
File IO Binair - Scanner – bv inlezen van het toetsenbord  
  
import java.util.\*;  
  
public class ScannerApp {  
 public static void main (String[] args) {  
 try ( Scanner scanner = new Scanner(System.in) {  
 System.out.println(“Please enter your name and age:”);  
 String name = scanner.next(); //leest tot spatie of nieuwe regels  
 int age = scanner.nextInt(); //getal lezen uit tekenreeks  
 System.out.printf(“Hello %s, you are %d old.”,name,age);  
 }  
 }  
}

ErrorStream  
  
setIn(); nieuwe inputstream kiezen  
setOut(); nieuwe outputstream kiezen  
setErr(); nieuwe errorstream kiezen  
  
import java.io.\*;  
  
public class ErrorApp {  
 public static void main (String[] args) {  
 try {  
 PrintStream logstream = new PrintStream(new FileOutputstream(“Logfile.txt”,true));  
 **System.setErr(logstream);**  
 }  
 catch (Exception ex) {  
 System.out.println(“Oops, something went wrong!”);  
 }  
 System.out.println(“Hello world”);  
 **System.err.println**(“This is an error message”);  
 }  
}  
  
File Handling  
  
import java.nio.Path;  
import java.nio.Paths;  
  
Path path = Paths.get(“C:\\temp\\newfile.txt”);  
  
**aanmaken:**  
  
try {  
 Files.createFile(path);  
} catch (IOException e) {  
e.printStackTrace();  
}  
  
**verwijderen:**  
  
try {  
 Files.deleteIfExists(path);  
} catch (IOException e) {  
e.printStackTrace();  
}  
  
**verplaatsen:**  
  
try {  
 Files.move(path, Paths.get(“C:\\temp\\movedfile.txt”));  
} catch (IOException e) {  
e.printStackTrace();  
}  
  
**verdubbelen:**  
  
try {  
 Files.copy(path, Paths.get(“C:\\temp\\movedfile.txt”));  
} catch (IOException e) {  
e.printStackTrace();  
}  
  
Exception Handling (*Checked* Exceptions – verplicht opvangen / *Unchecked RunTimeExceptions* – mag maar moet niet / *Unchecked Error* – nooit opvangen)  
  
**throw:**  
  
public void print (int copies)   
{  
 if (copies < 0 )  
 **throw new IllegalArgumentException**(“Can’t print less than 0 copies”);  
}  
  
**catch:**  
  
**try** **{**  
 printer.print(-1); voorbeeld van method in try

**} catch (**IllegalArgumentException **e) {**  System.out.println(e.getMessage());  
 **}**  
 **finally  
 {**  
 end methods of sluiters (zie bv Streams)  
 **}**  
  
**andere:**  
IOException dan bij catch e.printStackTrace();  
Exception  
FileNotFoundException  
InterruptedException  
  
We kunnen ook zelf nieuwe Exceptions Aanmaken in de catch om de Exception die we vangen verder door te smijten met maar dan specifieker:  
} catch (IOException e) {  
 throw new PrinterException(e , “Printer failed”);  
}  
  
  
Enumerations  
  
public static void main(String[] args)  
{  
Printer<ColorCartridge> printer=new Printer<ColorCartridge>(“MY PRINTER”,**ColorCartridge.BLUE**);  
textToPrint += **cartridge.printColor**();  
  
for (ColorCartridge cartridge : ColorCartridge.values())  
 {  
 System.out.println(cartridge.printColor() );  
 }  
  
}  
  
public interface ICartridge  
{  
 public String getFillPercentage();  
 public String printColor();  
}  
  
public **enum** ColorCartridge implements ICartridge  
{  
 **RED(“Red”),** op basis van de constructor extra veld **BLEU(“Blue”),** extra veld kan aangesproken worden **GREEN(“Green”),  
 YELLOW(“Yellow”);**  
  
**private ColorCartridge(String colorText)** ctrl + 1 maakt automatisch velden aan inconstructor ALTIJD PRIVATE voor ENUM **{  
 this.colorText = colorText;  
 }**  
 @Override  
 public String toString() toString overschrijven  
 {  
 return “Color!”;  
 }  
 @Override voorbeeld van extra toepassing  
 public String getFillPercentage()  
 {  
 return “97%”;  
 }  
 @Override  
 public String printColor(){  
 return colorText;  
 }  
}  
  
JAR file  
Aanmaken:  
Rechts klikken op het project.  
Export  
Type: Java/Runnable JAR file  
Launch type: test aanklikken  
locatie opgeven  
package required libraries into generated JAR  
  
Op te openen:  
Voor commandline app gewoon dubbelklikken.

Voor GUI in cmd naar de juiste folder gaan en dan volgende structuur gebruiken met de juiste filenaam.  
