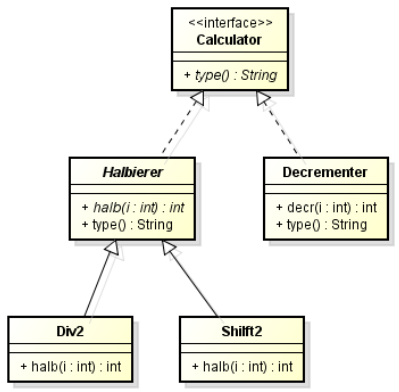
**PR2 – Formular für Lesenotizen**

**SS2021**

public interface Calculator {

String type();

}

public class Decrementer implements Calculator {

public int decr(int i) {

return i-1;

}

@Override public String type() {

return "dash calculation";

} }

public abstract class Halbierer implements Calculator {

@Override public String type() {

return "point calculation";

}

public abstract int halb(int i);

}

public class Div2 extends Halbierer {

@Override public int halb(int i) {

return i/2; }

}

public class Shift2 extends Halbierer {

@Override public int halb(int i) {

return i >> 1;

} }

@Override public int hashCode() {

return java.util.Objects.hash(s, c);

}

public class K {

private String s;

private char c;

// Konstruktor ausgeblendet und als vorhanden angenommen

// ...

public class Person **implements Comparable<Person>** {

**@Override public int compareTo(Person other) {**

**return Integer.compare(other.alter, alter);**

**}**

}

NullPointerException

@Override public boolean equals(Object o) {

if (o instanceof K) {

K k= (K)o;

boolean eq;

if (s == null) {

eq= (k.s == null);

} else {

eq= s.equals(k.s);

}

return (eq && c==k.c);

}

return false;

}

|  |  |  |  |
| --- | --- | --- | --- |
| Nachname  Lushaj | Vorname  Detijon | Matrikelnummer  1630149 | Abgabedatum:  08.06.21 |

public class MessageSourceMain {

public static void main(String[] args) {

print(

**// Hier fehlt Ihr Code.**

**new MessageSource() {**

**@Override public String getMessage() {**

**return "Hallo Welt";**

**}**

**}**

**() -> "Hallo Welt"**

**// Hier endet Ihr Code**

); }

public static void print(MessageSource src) {

System.out.println(src.getMessage());

} }

Programmieren Sie eine main-Methode, die einen zweiten Thread startet und auf dessen Beendigung wartet. Der zweite Thread soll „Thread 2“ auf der Console ausgeben. Die main- Methode soll nach dem Start des zweiten Threads und vor dem Warten auf Beendigung des zweiten Threads den Text „Main“ auf der Console ausgeben.

**public static void** main(String[] args) **throws** InterruptedException {

Thread t= **new** Thread() {

@Override **public void** run() {

System.*out*.println("Thread 2");

} };

t.start();

System.*out*.println("Main");

t.join();

}

Alternativen denkbar (Extra-Klasse, die Runnable implementiert / von Thread erbt, etc.).

page26image25223088page26image25227248Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

@Override public void start(Stage primaryStage) {

HBox root = new HBox();

Button btn1= new Button("Hü");

Button btn2= new Button("Hott");

btn2.setDisable(true);

btn1.setOnAction(new EventHandler<ActionEvent>() {

@Override public void handle(ActionEvent event) {

btn2.setDisable(false);

btn1.setDisable(true);

}

});  
 btn2.setOnAction(new EventHandler<ActionEvent>() {

@Override public void handle(ActionEvent event) {

btn1.setDisable(false);

btn2.setDisable(true);

}

});

root.getChildren().addAll(btn1, btn2);

Scene scene = new Scene(root);

primaryStage.setScene(scene);

primaryStage.show();

}

Wenn dieses Programm gestartet wird, beobachtet man entweder falsche Ergebnisse (die Summe der Zahlen von 1 bis 10.000 ist eigentlich 50.005.000, tatsächlich gibt das Programm aber häufig leicht niedrigere Werte aus), oder es bleibt einfach in einer Endlosschleife hängen, oder es bricht mit einer Exception ab.

Die beiden Threads P und C rufen voneinander unabhängig die Methoden push und pop der Instanz s auf. s verweist auf das gemeinsam genutzte Array list. Es kann vorkommen, dass P gerade dabei ist, die list zu verlängern (list= new int[size\*2]), während C in diesem Moment das 0-te Element der Liste lesen will (v=list[0]). C erntet dadurch manchmal v=0, wo ein größerer Wert richtig wäre.

long n= console.nextLong();

long cnt= LongStream.rangeClosed(1, n).filter( i -> n%i==0 ).count();

Die Anzahl der Teiler von n.

Was ist der Haupteinsatzzweck von Streams?   
Wenn die Berechnung auf mehrere Threads verteilt werden soll, ist dies mit Streams häufig eleganter möglich.

**Teilaufgabe b)** Wie muss das Programm verändert werden, um ein korrektes Verhalten zu erreichen. Nehmen Sie nur notwendige Veränderungen vor. Zu viele Veränderungen führen zu Punktabzug

Die beiden Methoden push und pop müssen als synchronized markiert werden. Die Methode available kann „normal“ bleiben. Da es nur einen C-Thread gibt, ist die folgende Sequenz KEIN kritischer Abschnitt und muss daher nicht extra mit synchronized gesichert

werden:

while (!s.available()) {

// busy wait

}

sum += s.pop();