PR2 – Formular für Lesenotizen

SS2021

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	08.06.21

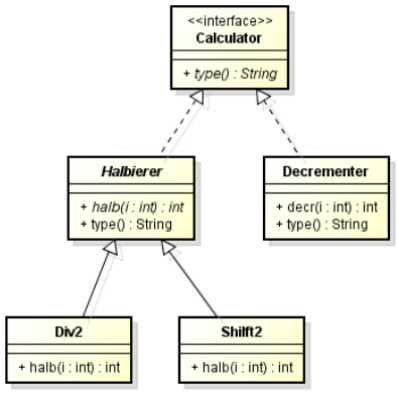
```
public interface Calculator {
  String type();
}
```

```
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;
} }
```




```
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;
    }
```

```
@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);
    }
}
```

```
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.).

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.

```
class S {
  private int[] list;
  private int size;
  public S() {
    list = new int[10];
    size = 0;
  }
  public void push(int v) {
    if (size == list.length) {
      int[] old = list;
      list = new int[size*2];
      for (int i = 0; i < size; i++) {
        list[i] = old[i];
      }
    }
    list[size++] = v;
  }

  public int pop() {
    if (size == 0) {
      throw new IllegalStateException();
    }
    int v = list[0];
    size--;
    for (int i = 0; i < size; i++) {
      list[i] = list[i+1];
    }
    return v;
  }

  public boolean available() {
    return size > 0;
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    S s = new S();
    new Thread(new P(s)).start();
    new Thread(new C(s)).start();
  }
}

class P implements Runnable {
  private S s;
  public P(S s) {
    this.s = s;
  }
  @Override public void run() {
    for (int i = 1; i <= 10000; i++) {
      s.push(i);
    }
  }
}

class C implements Runnable {
  private S s;
  public C(S s) {
    this.s = s;
  }
  @Override public void run() {
    long sum = 0;
    for (int i = 1; i <= 10000; i++) {
      while (!s.available()) {
        // busy wait
      }
      sum += s.pop();
    }
    System.out.println(sum);
  }
}
```

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();
```

```
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.

```
@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();
}
```