

Nur zum Selbststudium!

0.12 Aufgabe

Geben Sie die Lösungsworte der Quizze aus der Lernnotiz an.

38. Aufgabe (Exceptions analysieren)

```
public class Analyse {
    public int check(int x)
        throws MeineException {
        EinUndAusgabe io
            = new EinUndAusgabe();
        try {
            io.ausgeben("A");
            if (x < 20) {
                throw new MeineException(x);
            }
            io.ausgeben("B");
            if (x < 62) {
                io.ausgeben("C");
                return x;
            } else {
                io.ausgeben("D");
                throw new MeineException(x-10);
            }
        } catch (MeineException e) {
            try {
                io.ausgeben("E");
                if (e.getStufe() > 72) {
                    throw new MeineException(10);
                }
                io.ausgeben("F");
                if (e.getStufe() < 62) {
                    io.ausgeben("G");
                } else {
                    io.ausgeben("H");
                    throw new MeineException(x);
                }
            } finally {
                io.ausgeben("Y");
            }
        } finally {
            io.ausgeben("Z");
        }
        io.ausgeben("I");
        return -42;
    }
}
```

```
public class MeineException extends
Exception {

    private int stufe;

    public MeineException(int wert) {
        super();
        this.stufe = wert;
    }

    public int getStufe() {
        return this.stufe;
    }
}
```

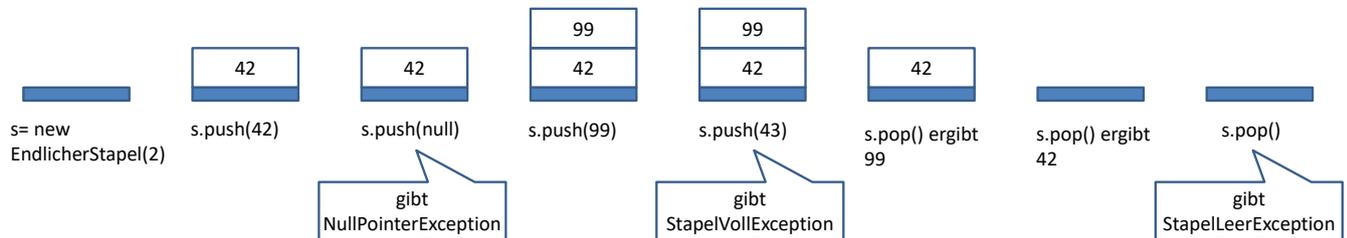
```
public class Main {

    public static void main(
        String[] args) {

        EinUndAusgabe io
            = new EinUndAusgabe();
        int[] test = {15, 35, 65, 75, 85};
        Analyse a = new Analyse();
        for(int t: test) {
            io.ausgeben(t + ": ");
            try {
                io.ausgeben(a.check(t) + "\n");
            } catch (Exception e) {
                io.ausgeben("Schicht\n");
            }
        }
    }
}
```

Geben Sie die Ausgaben des obigen Programms an.

39. Aufgabe (Stapel-Variante, Umgang mit Exceptions, Testen)



Ein Stapel (Stack) ist eine Datenstruktur, die Objekte eines bestimmten Typs aufnehmen kann. Dabei werden Objekte mit einer Methode `push(.)` so auf den Stapel gelegt, dass das neue Element immer oben liegt. Mit der Methode `pop(.)` wird das oberste Element zurückgegeben und vom Stapel gelöscht. Weiterhin ist es sinnvoll, wenn es Methoden `istLeer()` bzw. `istVoll()` gibt, mit denen man prüfen kann, ob ein Stapel voll oder leer ist.

- Realisieren Sie eine Klasse `EndlicherStapel` für Integer-Objekte, mit einem Konstruktor, der die mögliche Größe des Stapels als Parameter hat. *Nutzen Sie zur Realisierung des Stapels einen Array.*
- Realisieren Sie eine Methode `push(Integer)`, mit der ein Integer-Objekt auf den Stapel gelegt wird. Wird versucht, eine null-Referenz auf den Stapel zu legen, soll eine `NullPointerException` geworfen werden. Wird versucht ein Objekt auf einen vollen Stapel zu legen, soll eine von Ihnen neu zu erstellende `StapelVollException` geworfen werden.
- Realisieren Sie die Methode `pop()`, die das zuletzt auf den Stapel gelegte Integer-Objekt zurück gibt und vom Stapel löscht. Wird versucht, ein Objekt von einem leeren Stapel zu nehmen, soll eine von Ihnen neu zu erstellende `StapelLeerException` geworfen werden.
- Realisieren Sie die Methoden `istVoll()` und `istLeer()` zur Überprüfung des Stapelzustands.
- Schreiben Sie zu allen realisierten Methoden JUnit-Tests, die alle möglichen Ergebnisse testen. Nutzen Sie dazu eine Test Fixture, also mehrere Objekte, mit einem leeren, einem vollen und einem „normal“ gefüllten Stack, auf denen Sie jeweils alle Ihre Methoden ausprobieren.
- Kopieren Sie Ihr bisheriges Projekt in ein neues Projekt und ändern Sie Ihre Implementierung so ab, dass alle Tests aus e) erfüllt werden, sich aber trotzdem ein Fehler im Programm befindet. Ergänzen Sie dann einen Test, der den eingebauten Fehler findet.

Hinweis: Die Methoden `push(.)` und `pop()` sollen die Exceptions nur werfen, nicht selber bearbeiten. In den Tests ist mit `try-catch`-Blöcken zu prüfen, ob die richtigen Exceptions geworfen werden und die Methoden sonst die korrekten Ergebnisse liefern.

Quiz

Quiz 1 Quiz 2 Quiz 3
E X C

Hilfe

Quiz 1: Exception 1

Welche der folgenden Aussagen sind korrekt?

- FileNotFoundException ist eine normale Java-Klasse.
- Jede Exception, die eine Methode werfen könnte, muss in der throws-Liste der Methode stehen.
- Jede Methode kann nur einmal einen try-Block haben.
- try-Blöcke können ineinander geschachtelt vorkommen.
- Eine NullPointerException kann nicht gefangen werden.

Antwort überprüfen

Quiz 2: Exception 2

Welche der folgenden Aussagen sind korrekt?

- Zu jedem try-Block muss es einen catch-Block geben.
- Zu jedem try-Block muss es einen finally-Block geben.
- Innerhalb eines catch-Blocks kann keine neue Exception geworfen werden.
- Innerhalb eines finally-Blocks kann keine neue Exception geworfen werden.
- Eine Exception kann nicht gleichzeitig in einer throws-Liste einer Methode und in einem catch-Block der gleichen Methode stehen.

Antwort überprüfen

Quiz 3: Exception 3

Welche der folgenden Aussagen sind korrekt?

- Exceptions deuten immer auf Programmierfehler hin.
- Die Reihenfolge der catch-Blöcke eines try-Blocks spielt keine Rolle.
- Innerhalb eines finally-Blocks kann kein neuer try-Block stehen.
- Innerhalb eines finally-Blocks kann keine lokale Variable deklariert werden.
- Ein try-Block kann mehrere finally-Blöcke haben.

Antwort überprüfen

zu 38) (prog1AufgabeExceptionAnalyse)
15: AEFYZI-42
35: ABCZ35
65: ABDEFGYZI-42
75: ABDEFHYZSchicht
85: ABDEYZSchicht

39 (AufgabeStackLoesung) a) -d) (hier konsequent Integer genutzt, int generell auch erlaubt)

```
public class EndlicherStapel{

    private int position=0;
    private Integer[] stapel; // auch int[] erlaubt

    public EndlicherStapel(Integer anzahl){
        this.stapel = new Integer[anzahl];
    }

    public void push(Integer i) throws StapelVollException{
        if (i==null){
            throw new NullPointerException();
        }
        if (this.position>=stapel.length){
            throw new StapelVollException();
        }
        this.stapel[this.position]=i;
        this.position = this.position+1;
    }

    public Integer pop() throws StapelLeerException{
        if(this.position == 0){
            throw new StapelLeerException();
        }
        this.position = this.position-1;
        return this.stapel[this.position];
    }

    public Boolean istLeer(){
        return this.position == 0;
    }

    public Boolean istVoll(){
        return this.position>=stapel.length;
    }

}
```

e)

```
import org.junit.Assert;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class EndlicherStapelTest{

    private EndlicherStapel normal;
    private EndlicherStapel leer;
    private EndlicherStapel voll;
```

```
@Before
public void setUp() throws StapelVollException, StapelLeerException{
    this.normal = new EndlicherStapel(5);
    this.normal.push(42);
    this.leer = new EndlicherStapel(4);
    this.voll = new EndlicherStapel(3);
    this.voll.push(42);
    this.voll.push(43);
    this.voll.push(44);
}

@Test
public void testIstLeer1(){
    Assert.assertTrue(leer.istLeer());
}

@Test
public void testIstLeer2(){
    Assert.assertTrue(!voll.istLeer());
}

@Test
public void testIstLeer3(){
    Assert.assertTrue(!normal.istLeer());
}

@Test
public void testIstVoll1(){
    Assert.assertTrue(!this.leer.istVoll());
}

@Test
public void testIstVoll2(){
    Assert.assertTrue(this.voll.istVoll());
}

@Test
public void testIstVoll3(){
    Assert.assertTrue(!this.normal.istVoll());
}

@Test
public void testpop1(){
    try{
        this.leer.pop();
        Assert.fail();
    } catch (StapelLeerException e){
    }
}

@Test
```

```
public void testpop2(){
    try{
        Assert.assertTrue(this.voll.pop().equals(44));
    } catch (StapelLeerException e){
        Assert.fail();
    }
}

@Test
public void testpop3(){
    try{
        Assert.assertTrue(this.normal.pop().equals(42));
    } catch (StapelLeerException e){
        Assert.fail();
    }
}

@Test
public void testpush1(){
    try{
        this.leer.push(null);
        Assert.fail();
    } catch (NullPointerException e){
    } catch (StapelVollException e){
        Assert.fail();
    }
}

@Test
public void testpush2(){
    try{
        this.voll.push(43);
        Assert.fail();
    } catch (StapelVollException e){
    }
}

@Test
public void testpush3(){
    try{
        this.normal.push(99);
        Assert.assertTrue(this.normal.pop().equals(99));
    } catch (StapelLeerException e){
        Assert.fail();
    } catch (StapelVollException e){
        Assert.fail();
    }
}

@Test // für f) ergänzt
public void testMerkwuerdigesPush(){
    try{
```

```
        this.normal.push(88);
        Assert.assertTrue(this.normal.pop().equals(88));
    } catch (StapelLeerException e){
        Assert.fail();
    } catch (StapelVollException e){
        Assert.fail();
    }
}
}
```

zu f)

```
public void push(Integer i) throws StapelVollException{
    if (i==null || i.equals(88)){ // plumper Fehler
        throw new NullPointerException();
    }
    if (this.position>=stapel.length){
        throw new StapelVollException();
    }
    this.stapel[this.position]=i;
    this.position = this.position+1;
}
```