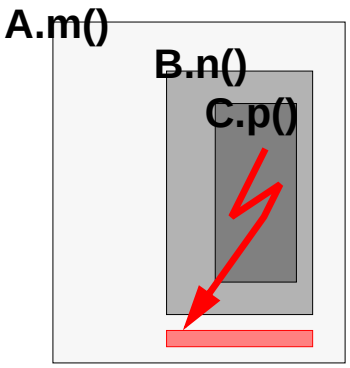
Z.11 Fehlerbehandlung

- Programm beenden (`System.exit()`)
  - meist eine schlechte Idee
- Ausgabe einer Fehlermeldung
  - hilft nicht den Fehler zu überwinden
- spezieller Rückgabewert kennzeichnet Fehler
  - Konstruktoren haben keinen Rückgabewert
  - Was ist wenn die Methode den Wertebereich des Rückgabewerts bereits voll ausnutzt?
- Aufruf einer benutzerdefinierten Fehlerroutine
  - unschön
  - Was muss diese Methode tun?
- Lösung: Ausnahmebehandlung (Exceptions)!

1 Was ist Ausnahmebehandlung?

- Weiterreichen des Programmflusses vom Fehlerursprung zur Fehlerbehandlung



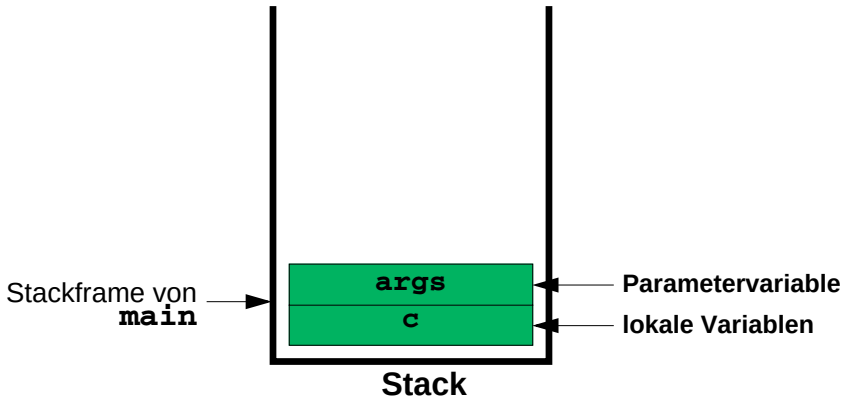
- Verantwortlichkeit:
  - Autor eines Codestücks kann den Fehler erkennen, weiß jedoch nicht wie er behandelt werden soll.
  - Benutzer des Codes weiß was zu tun ist, hat jedoch nicht die Möglichkeit den Fehler zu erkennen.

2 Was passiert bei einem Methodenaufruf?

```
class Customer {
 void createAccount
                (Bank bank) {
  Account account =
                new Account();
  bank.newAccount(account, 5);
```

```
class Bank {
  void newAccount
         (Account a, int i) {
        int counter = 0;
    ...
  }
```

```
class Main {
  public static void main(String
args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
```

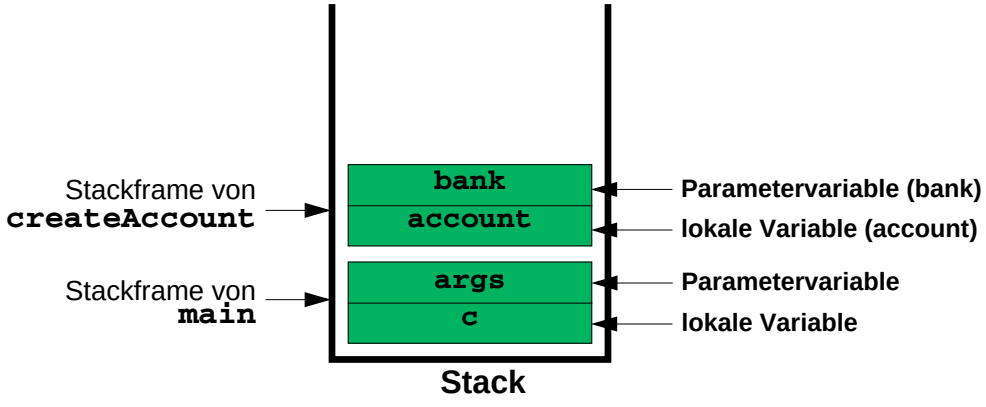


2 Was passiert bei einem Methodenaufruf? (2)

```
class Customer {
 void createAccount
                (Bank bank) {
  Account account =
                new Account();
  bank.newAccount(account, 5);
```

```
class Bank {
  void newAccount
         (Account a, int i) {
        int counter = 0;
    ...
  }
```

```
class Main {
  public static void main(String
args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
```

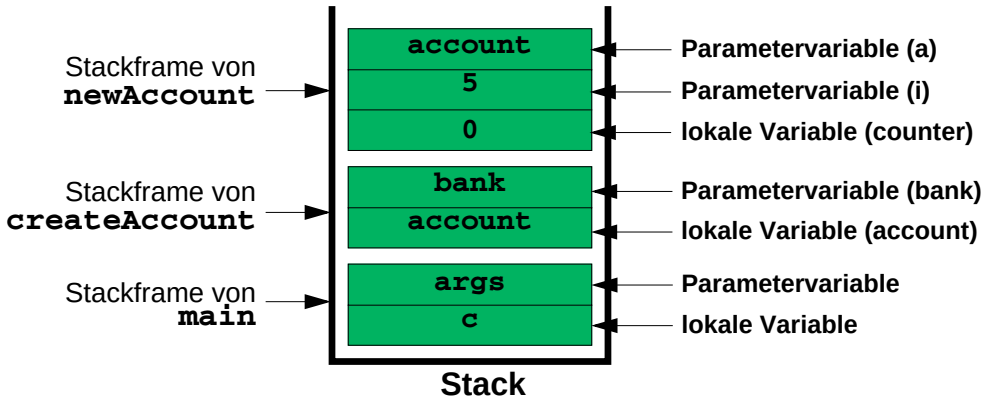


2 Was passiert bei einem Methodenaufruf? (3)

```
class Customer {
 void createAccount
                (Bank bank) {
  Account account =
                new Account();
  bank.newAccount(account, 5);
```

```
class Bank {
  void newAccount
         (Account a, int i) {
        int counter = 0;
    ...
  }
```

```
class Main {
  public static void main(String
args[]) {
        Customer c = new Customer();
        c.createAccount(new Bank());
```



3 Try, Throw und Catch-Anweisung

```
try {
    ...
    if (...) throw new MyException();
    ...
} catch(MyException e) {
    // exception handler
    ...
}
```

- throw wird benutzt um eine *Ausnahme* (*Exception*) zu werfen
- ein **`catch`**-Block muss direkt nach dem **`try`**-Block folgen
- es kann mehr als nur einen **`catch`**-Block geben
  - der passende catch-Block wird nach der Programmreihenfolge gesucht
- ein Methode muss nicht alle Exception fangen
  - nicht gefangene Exception werden automatisch an die aufrufende Methode weitergereicht

4 Finally-Anweisung

- der **`finally`**-Block wird immer beim Verlassen eines **`try`**-Blockes ausgeführt
  - ◆ kann zum Aufräumen benutzt werden bei (un-)behandelten Ausnahmen

```
try {
    ...
} catch(...) {
    ...     // Fehlerbehandlung
} finally {
    ...     // Ressourcen freigeben
}
```

- ein **`finally`**-Block kann auch ohne **`catch`**-Block benutzt werden:

```
try {
    ...
    if (...) return;
    ...
} finally { ... }
```

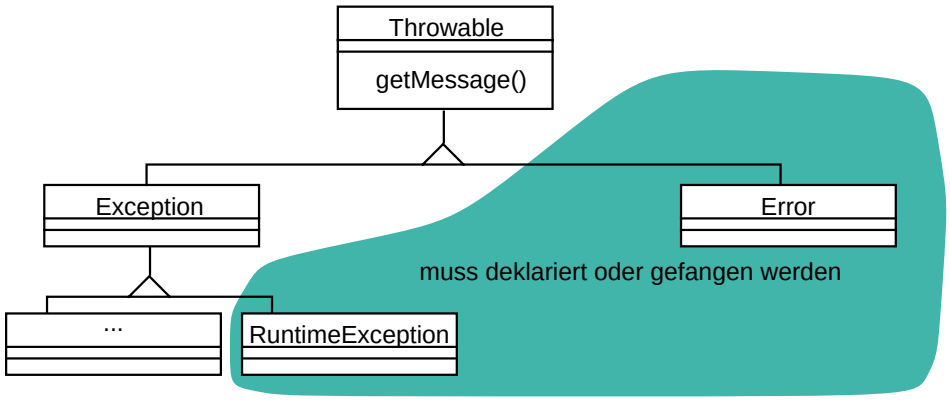

5 throws-Anweisung

- unbehandelte Exceptions müssen bei der Methode angeben werden:

```
class Test {
    void m() throws MyException {
        ...
        if (...) throw new MyException();
        ...
    }
}
```

6 Fehlerklassen

- alle Exceptions sind von **`Throwable`** abgeleitet
- Ausnahmen die beinahe überall auftreten können sind:
  - **`Error`**: Linkerfehler, Fehler im Format von Klassendateien, out of memory,..
  - **`RuntimeException`**: array index, null pointer, illegal cast, ...
- Ausnahmen von Anwendungen sind von **`java.lang.Exception`** abgeleitet



7 Ausnahmen und Vererbung: Fehlerbehandlung

- Ausnahmebehandlung von Unterklassen durch mehrere catch-Blöcke
- Bemerkung: Die Oberklasse behandelt alle Unterklassen, die Oberklasse sollte daher immer am Ende “gefangen” werden:

```
class MathException {}
class ZeroDivideException extends MathException {}
class InvalidArgException extends MathException {}
try {
    ...
} catch(ZeroDivideException e) {
    ...
} catch(InvalidArgException e) {
    ...
} catch(MathException e) {
    ...
}
```

8 Beispiel

```
class TestException extends Exception {
  public TestException(String s) {super(s);}
}

public class Test {

  public void hello() throws TestException {
    if (...) throw new TestException("...an error msg...");
  }

  public void testIt() {
    try {
      hello();
      ...
    }
    catch (TestException t) {
      System.out.println("Exception raised:" + t.getMessage());
    }
    finally {
      // clean up
    }
  }
}
```


9 Ausnahmen und Vererbung: Throwing

- Können überschriebene Methoden andere Exceptions werfen, als die ursprüngliche Methode?
- Grundsatz:
  - Unterklassen können überall dort verwendet werden wo die Oberklasse erwartet wird.
  - Unterklassen sind “besser” als Oberklassen.
- das bedeutet:
  - Unterklassen dürfen keine zusätzlichen Exception werfen.
  - Unterklassen können Unterklassen von den deklarierten Ausnahmen der Oberklasse werfen.
  - Unterklassen dürfen keine Oberklassen von den ursprünglich deklarierten Ausnahmen werfen.

9 Beispiel

```
class E1 extends Exception {}
class E2 extends Exception {}
class E3 extends E2 {}
class A {
    void m() throws E2 {}
}
class B extends A {
    void m() throws ??? {}
}
```

- ??? =

9 Beispiel (2)

```
class E1 extends Exception {}
class E2 extends Exception {}
class E3 extends E2 {}
class A {
    void m() throws E2 {}
}
class B extends A {
    void m() throws ??? {}
}
```

- ??? =

Falsch sind:	Richtig sind:
E1	**E2**
Exception	**E3**
...	keine

10 Zusammenfassung: Exceptions

- werfen von Ausnahmen: `throw new MyException("...");`
- Programm-Block für den die Ausnahmebehandlung gilt: `try { ... }`
- Ausnahmebehandlung:

```
try {
... throw new MyException("..."); ...
} catch(MyException e) { ... }
```

- Zusätzlich: **`finally`**-Block
- Ausnahmen müssen von **`Throwable`** abgeleitet sein.
- Ausnahmen von Anwendungen sollten von **`Exception`** abgeleitet werden.
- Ausnahmen müssen bei der Methodendeklaration angegeben werden:

```
void mymethod() throws ...
```