



Informatik I

Übungsstunde 13

Herbst 2020

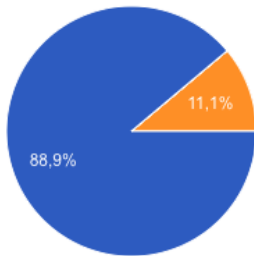
Fragen

Fragen?

Repetition

Ich würde bevorzugen, in der letzten Stunde:

9 Antworten

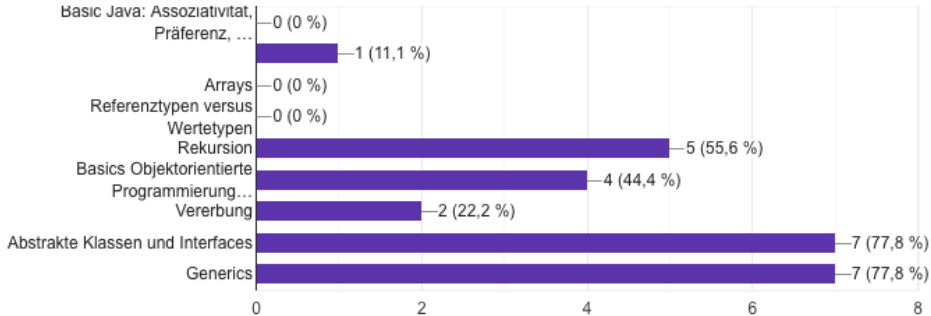


- Themen zu repetieren und Prüfungsfragen anzuschauen
- Nur Themen zu repetieren
- Nur Prüfungsfragen anzuschauen
- Egal 🤷

Repetition

Themen: Wählt alle Themen aus, die nützlich zum Repetieren wären.

9 Antworten



Generics

Auf Deutsch: Generisch, allgemein, Allgemeinheit

Generics

Generics ermöglichen es, dass Typen (Klassen und Interfaces) Parameter bei der Definition von Klassen, Interfaces und Methoden sind.

Generics

1, "abc" @ae56

Ähnlich wie die bekannteren formalen Parameter, die in Methodendeklarationen verwendet werden, bieten Typparameter eine Möglichkeit, denselben Code mit unterschiedlichen Eingaben wiederzuverwenden.

String.class
LinkedList.class

Generics

Der Unterschied ist, dass die Eingaben für formale Parameter Werte sind, während die Eingaben für Typparameter Typen sind.

Generics

"generisch" bezieht sich also auf den Typ. Wir implementieren unseren Code (Klassen) für einen beliebigen (generischen, allgemeinen) Typ.

Java Generics

Standardbibliothek von Java stellt einige sehr nützliche Klassen zur Verfügung.

Beispiel: ArrayListe. Array-Listen verhalten sich (fast) so wie Arrays, sie können Ihre Grösse jedoch zur Laufzeit ändern. Dies ist mit Klassen implementiert.

Damit nicht für jeden Basistyp eine eigene Klasse implementiert werden muss, gibt es in Java **Generics**.

```
ArrayList<Temperature> list; // an empty list
```

```
list.add(new Temperature("Zurich","12.11.2020",12)); // list grows ...  
list.add(new Temperature("Bern","17.11.2020",9)); // ... on demand
```

Typ Parameter („Parametrischer Polymorphismus“)

In Java kann man eine Klasse mit einem Typ parametrisieren

```
class Pair <T> { // Paar mit generischem Werttyp T
    private T first;
    private T second;
}
```

Platzhalter T

```
Pair(T first, T scnd){
    first = first;
    second = scnd;
}
}
```

Konkreter Typ String wird für T in Pair eingesetzt.

Verwendung:

```
Pair<String> s = new Pair<String>("ETH", "Zurich");
```

Pair API (Programmierschnittstelle)

```
class Pair<T> {  
    ...  
    // construct a pair consisting of first and scnd  
    Pair(T first, T scnd)  
    // return first element  
    public T getFirst();  
    // return second element  
    public T getSecond();  
    // exchange first and second element  
    public void swap();  
    // return a string comprising the two elements  
    public String toString();  
}
```

API = Application Programming Interface

Motivation Generics

Code Beispiel (CodeExpert)

Pair Class



Paare von Zahlen

int: unboxed type; Integer: Boxed type Float.

- Java Generics können nur auf Objekten operieren
- Fundamentaltypen `int`, `float` (etc.) sind keine Objekte
- Verwende Wrapperklassen für Fundamentaltypen an, z.B. den Typ **Integer**
- Java macht *autoboxing* und packt einen Fundamentaltyp automatisch in eine Wrapperklasse ein, wo nötig.

~~int~~

```
Pair<Integer> i = new Pair<Integer>(3,5);  
int a = i.getFirst(); // auto unboxing: Integer -> int
```

Paare von Paaren?

Paare von Paaren?

Kein Problem:

```
Pair<Integer> p1 = new Pair<Integer>(2,3);  
Pair<Integer> p2 = new Pair<Integer>(42,2);  
Pair<Pair<Integer>> p = new Pair<Pair<Integer>> (p1, p2);  
Out.println(p); // ((2,3),(42,2))
```

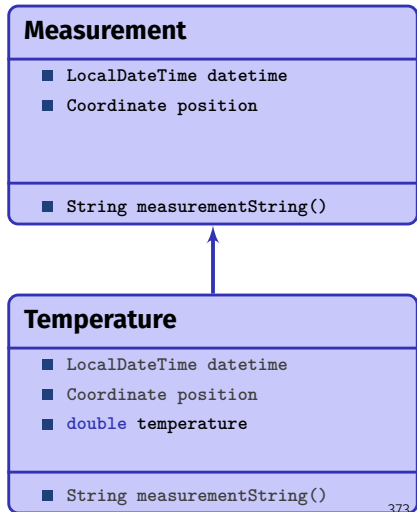


Abstrakte Klassen und Interfaces

Abstrakte Klassen. Beispiel Vorlesung: Measurement,
Temperature, Wind.

Vererbung: extends Schlüsselwort

Messung

```
class Measurement {  
    LocalDateTime datetime;  
    Coordinate position;  
  
    String measurementString() {...}  
}  
  
class Temperature extends Measurement {  
    double temperature;  
}  
  
class Wind extends Measurement {  
    double speed;  
    double direction;  
}
```



Abstrakte Klassen

```
class Measurement {  
    ...  
    // returns 'true' if measurement is alarming, 'false' otherwise  
    public boolean alarm() {...}  
}
```

- Klasse **Measurement** bietet eine Methode **alarm()** an
- Die Methode soll **true** zurückgeben, genau dann wenn die Messung **alarmierend** ist

Abstrakte Klassen

```
class Measurement {  
    ...  
    // returns 'true' if measurement is alarming, 'false' otherwise  
    public boolean alarm() {...}  
}
```

- Klasse **Measurement** bietet eine Methode **alarm()** an
- Die Methode soll **true** zurückgeben, genau dann wenn die Messung **alarmierend** ist
- ... aber die Methode macht eigentlich nur für die Subtypen Sinn

Abstrakte Klassen

Es macht keinen Sinn, Objekte vom Typ **Measurement** zu erstellen. Der Datentyp sollte **abstrakt** sein.

Abstrakte Klassen: Keyword abstract

```
abstract class Measurement {  
    ...  
    // returns 'true' if measurement is alarming, 'false' otherwise  
    abstract boolean alarm();  
}  
  
class Temperature extends Measurement {  
    double temperature;  
  
    // Implement the abstract method from the supertype  
    boolean alarm(){  
        return temperature > 35;  
    }  
}
```

Abstrakte Klassen: Keyword abstract

```
abstract class Measurement {  
    ... Date date;  
    // returns 'true' if measurement is alarming, 'false' otherwise  
    abstract boolean alarm();  
}
```

```
class Wind extends Measurement {  
    double speed;  
  
    // Implement the abstract method from the supertype  
    boolean alarm(){  
        return speed > 80;  
    }  
}
```

Abstrakte Klassen: Eigenschaften

- Falls mindestens eine Methode **abstract** ist, d.h. nicht implementiert, muss die ganze Klasse **abstract** deklariert sein.
- Abstrakte Klassen können **nicht** instanziiert werden (**new** ...)
- Abstrakte Klassen enthalten Daten und Code, welche von allen Subklassen geerbt wird. Von den Unterschieden wird abstrahiert.

Frage: Gibt es abstrakte Variablen?

Abstrakte Klassen und Interfaces

Interfaces

Interfaces

Ein Interface (übersetzt: Schnittstelle) definiert Funktionalität einer potentiellen Implementation durch eine Klasse

```
public interface Comparable<T>
{
    public int compareTo (T o);
}
```

1. Nur Methoden

2. Alle Methoden abstrakt
⇔ keine Implementation.

Jede Klasse **T**, welche **Comparable<T>** implementiert, muss die Methoden des Interfaces **Comparable<T>** anbieten.

```
public class Integer implements Comparable<Integer>{
    // contains this
    public int compareTo(Integer o){...}
}
```

Sortierte Paare?

```
class Pair<T> {  
    ...  
    public void swap(){  
        T temp = first;  
        first = second;  
        second = temp;  
    }  
  
    public void sort(){  
        if (second < first) {  
            swap();  
        }  
    }  
}
```

number, byte, String

Sortierte Paare?

```
class Pair<T> {  
    ...  
    public void swap(){  
        T temp = first;  
        first = second;  
        second = temp;  
    }  
  
    public void sort(){  
        if (second < first) {  
            swap();  
        }  
    }  
}
```

error: bad operand types for binary operator '<'

Besser?


```
class Pair<T> {  
    ...  
    public void swap(){  
        T temp = first;  
        first = second;  
        second = temp;  
    }  
  
    public void sort(){  
        if (first.compareTo(second) > 0) {  
            swap();  
        }  
    }  
}
```

Wieso error?

Besser?

```
class Pair<T> {  
    ...  
    public void swap(){  
        T temp = first;  
        first = second;  
        second = temp;  
    }  
  
    public void sort(){  
        if (first.compareTo(second) > 0) {  
            swap();  
        }  
    }  
}
```

error: cannot find symbol
compareTo



Sortiertes Paar!

```
class Pair <T extends Comparable<T>>{  
    ...  
  
    public void sort(){  
        if (first.compareTo(second) > 0) {  
            swap();  
        }  
    }  
    ...  
}
```

Sortiertes Paar!

```
class Pair <T extends Comparable<T>>{  
    ...  
  
    public void sort(){  
        if (first.compareTo(second) > 0) {  
            swap();  
        }  
    }  
    ...  
}
```

 extends Comparable<T> stellt sicher,
dass die Methode T.compareTo existiert.

```
Pair<Integer> p = new Pair<Integer>(8,3);  
p.sort();  
Out.println(p); // (3,8)
```


Paare vergleichen ?

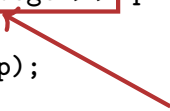
```
Pair<Integer> p1 = new Pair<Integer>(10,3);  
Pair<Integer> p2 = new Pair<Integer>(8,1);  
  
Pair<Pair<Integer>> p = new Pair<Pair<Integer>>(p1,p2);  
  
Out.println(p);  
p.sort();  
Out.println(p);
```

Paare vergleichen ?

```
Pair<Integer> p1 = new Pair<Integer>(10,3);  
Pair<Integer> p2 = new Pair<Integer>(8,1);
```

```
Pair<Pair<Integer>> p = new Pair<Pair<Integer>>(p1,p2);
```

```
Out.println(p);  
p.sort();  
Out.println(p);
```



error: type argument
Pair<Integer> is not within
bounds of type-variable T

Vergleichbare Paare

```
class Pair<T extends Comparable<T>> implements Comparable<Pair<T>>{  
    ...  
  
    // returns pair comparison with priority on first  
    public int compareTo(Pair<T> other){  
        if (this.first.equals(other.second)){  
            return this.second.compareTo(other.second);  
        } else  
            return this.first.compareTo(other.first);  
    }  
}
```

Vergleichbare Paare

```
Pair<Integer> p1 = new Pair<Integer>(10,3);  
Pair<Integer> p2 = new Pair<Integer>(8,1);  
  
Pair<Pair<Integer>> p = new Pair<Pair<Integer>>(p1,p2);  
  
Out.println(p); // ((10,3),(8,1))  
p.sort();  
Out.println(p); // ((8,1),(10,3))
```

Interfaces und Mehrfachvererbung in Java

Klassen können in Java nur von einer Klasse erben (eine Klasse erweitern)
– es gibt also **keine Mehrfachvererbung** – aber Klassen können **mehrere Interfaces** implementieren.



Klassen: ✓

Interfaces: X

Abstrakte Klassen vs. Interfaces

Was ist nun der Unterschied zwischen abstrakten Klassen und Interfaces?

Abstrakte Klassen vs. Interfaces

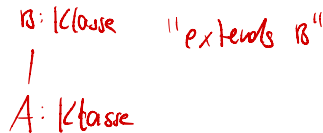
Was ist nun der Unterschied zwischen abstrakten Klassen und Interfaces?

- abstrakte Klassen können nicht-abstrakte Methoden besitzen, Interfaces können nur abstrakte Methoden besitzen.

Abstrakte Klassen vs. Interfaces

Was ist nun der Unterschied zwischen abstrakten Klassen und Interfaces?

- abstrakte Klassen können nicht-abstrakte Methoden besitzen, Interfaces können nur abstrakte Methoden besitzen.
- (abstrakte) Klassen besitzen **Methoden und Variablen**, Interfaces besitzen nur **Methoden**.



Abstrakte Klassen vs. Interfaces

Was ist nun der Unterschied zwischen abstrakten Klassen und Interfaces?

- abstrakte Klassen können nicht-abstrakte Methoden besitzen, Interfaces können nur abstrakte Methoden besitzen.
- (abstrakte) Klassen besitzen **Methoden und Variablen**, Interfaces besitzen nur **Methoden**.
- Wir können nur von **einer** (abstrakten) Klasse erben, aber von **mehreren** Interfaces.

Abstrakte Klassen vs. Interfaces

	(abstrakte) Klasse	Interfaces
Abstrakte Methoden erlaubt?	Ja	Ja
Nicht-abstrakte Methoden erlaubt?	Ja	Nein
Variablen erlaubt?	Ja	Nein
Kinder können mehrere erben?	Nein	Ja
Mehrfachvererbung?	Ja	Nein
Generisch?	Ja	Ja

A_1
|
 A_2
|
 A_3

Prüfungsaufgabe

- Vererbung und Polymorphie
- Java Collections (falls genug Zeit)

Ice-breaker - Booleans

and
true && false → false

true || false → true
or

Ice-breaker - Booleans

Heute ist ein Samstag und es regnet. Soll ich wandern gehen?

```
bool shouldHike = isWeekend && (!isRaining)
```

Ich bin 1m74 und ich bin nicht schwanger. Sollte ich in den Vergnügungspark gehen?

```
bool accessAmusementParkGranted = (height > 1m20) &&  
(!pregnant)
```

Ice-breaker - Booleans

```
bool accessGranted = isSuperAdmin || isUser &&  
    !isBlocked
```

$((a \cdot b) \cdot c)$

In diesem Szenario werden Super-Admins nicht als Benutzer betrachtet.

Ich bin ein Superadmin, kann ich auf die Website zugreifen?

Ich bin ein Benutzer und bin nicht blockiert, kann ich auf die Website zugreifen?

Ich bin ein Super-Admin, aber ich bin blockiert, kann ich auf die Website zugreifen?

→ Kannst du eine Sicherheitslücke erkennen? Wenn ja, wie würdest Du den Ausdruck ändern, um ihn zu beheben?

Pause bis 17:15

Ice-breaker - Arrays

Wie würdest du das maximale Element eines Arrays von ganzen Zahlen finden?

Zum Beispiel, für das Array $\{1, 4, -3, 5\}$, sollte das Ergebnis 5 sein.

Ice-breaker - Arrays

Wie würdest du die Summe der Elemente eines Arrays von ganzen Zahlen berechnen?

Zum Beispiel, für das Array $\{1, 4, -3, 5\}$ sollte das Ergebnis 7 sein.

Ice-breaker - Recursion

Wie würdest du $1 + 2 + 3 + \dots + n$ mittels Rekursion berechnen?

Ice-breaker - Recursion

Wie würdest du $1 * 2 * 3 * \dots * n = n!$ mittels Rekursion berechnen?

Ice-breaker - Classes

Du besitzt eine Werkstatt. Du verkaufst verschiedene Arten von Fahrzeugen: Lastwagen, Autos und Motorräder.

Jedes Fahrzeug hat eine Marke, einen Preis und eine Anzahl von Kilometern (0, wenn das Fahrzeug neu ist).

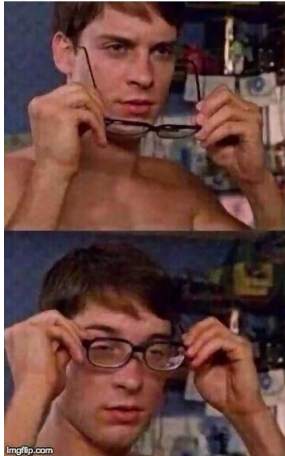
LKWs haben Informationen darüber, wie viele Gegenstände sie transportieren können.

Autos haben Informationen darüber, wie viele Passagiersitze sie haben.

Motorräder haben einen Index, wie cool sie aussehen (von 10).

Stelle deine Garage mit Klassen dar. Sei vorsichtig mit den Typen (`int`, `float`, etc).

Viel Erfolg :)



MERRY/XMAS

MANY EXAMS