# Java Fundamentals

## Generics (Generische Klassen und Typparameter)

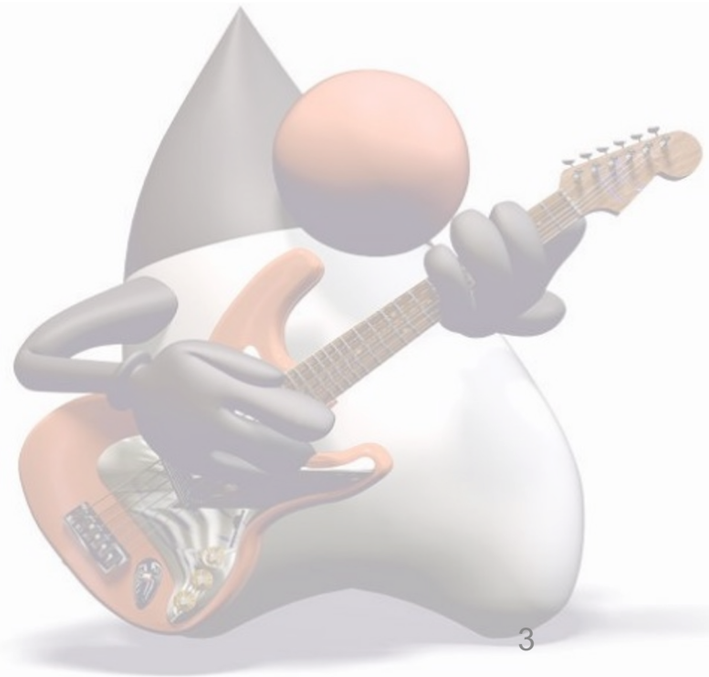
# Generics

- Generics werden verwendet um Klassen mit anderen Typen zu parametrisieren
- Generics wurden mit Java 5 eingeführt
- Vor Java 5 konnte man bei Datencontainern nur Objects speichern. Damit war die Typsicherheit vermindert.

# Generics

Vor Java 5 waren beispielsweise Collections nicht typsicher.

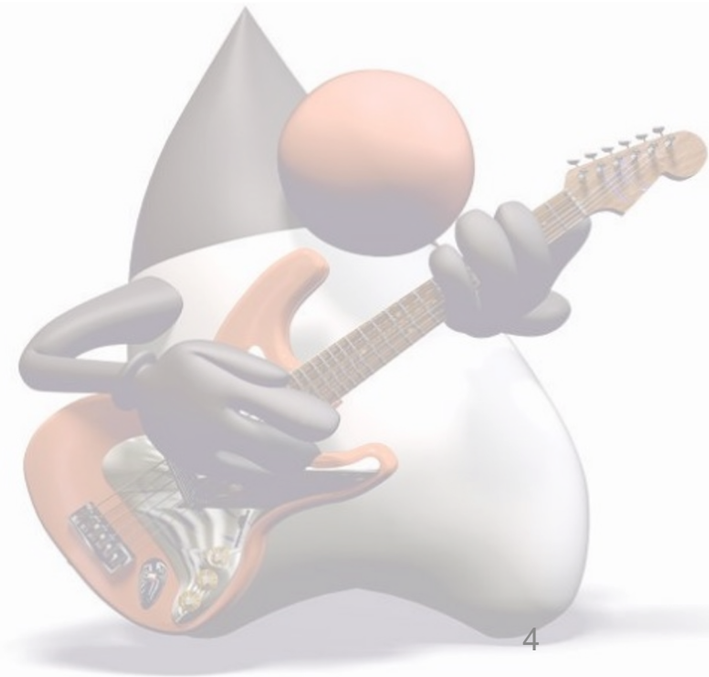
```
public class VorJava5 {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add("Eins");
        list.add(2);

        Object o = list.get(0); //get liefert Object
        String listElement;

        if (o instanceof String){
            listElement = (String)o;
        } else if (o instanceof Integer){
            listElement = ((Integer)o).toString();
        }
    }
}
```



# Generics

Ab Java 5 sind Collections typsicher

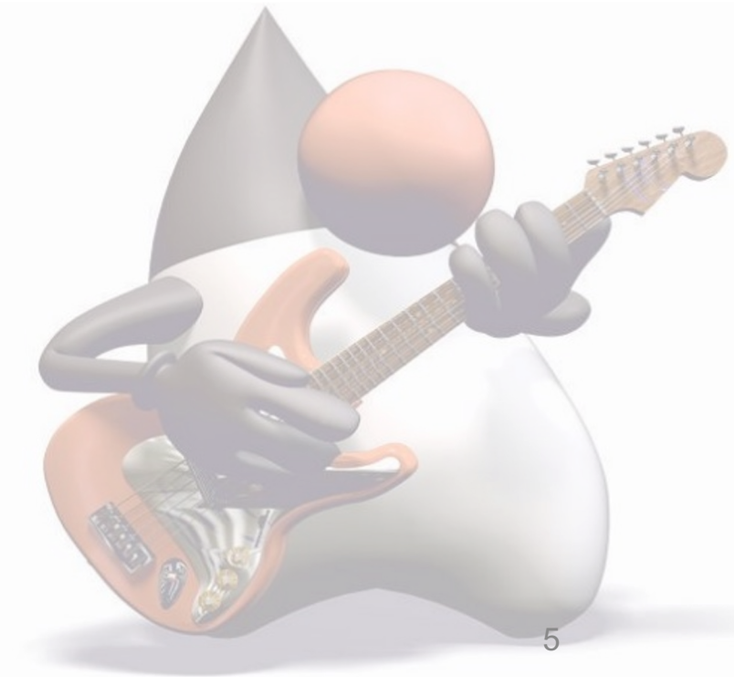
```
public class SeitJava5 {
    public static void main(String[] args) {
        List<String> aList = new ArrayList<>();
        aList.add("Eins");
        //aList.add(2); Compile Fehler

        String element = aList.get(0);
    }
}


//Interface List
public interface List<E> extends Collection<E> {
...
}
```



# Generics

Wir können generische Klassen selbst programmieren

```
public class Container<T> {
    T content;

    public Container(T content) {
        this.content = content;
        // new T()  -> Compile Fehler
    }

    public T getContent() {
        return content;
    }

    public void setContent(T content) {
        this.content = content;
    }
}
```



# Generics

Wir können generische Klassen selbst programmieren

```
public static void main(String[] args) {
    Container<String> stringContainer = new Container<>("Michael");
    String s = stringContainer.getContent();

    Container<Integer> intContainer = new Container<>(5);
    Integer i = intContainer.getContent();
}
```

# Generics

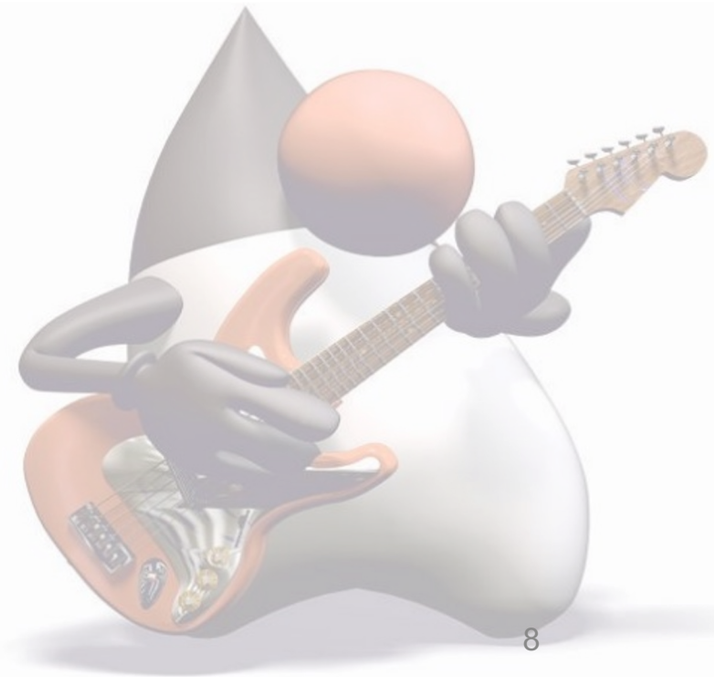
Die Typparameter können mit bounds versehen werden

```
public class NumberContainer<T extends Number> {
    private T number;

    public NumberContainer(T number) {
        this.number = number;
    }

    public T getNumber() {
        return number;
    }

    public void setNumber(T number) {
        this.number = number;
    }

    public long getIntValue(){
        return number.longValue();
    }

    public double getDoubleValue(){
        return number.doubleValue();
    }
}
```

# Generics

Die Typparameter können mit bounds versehen werden

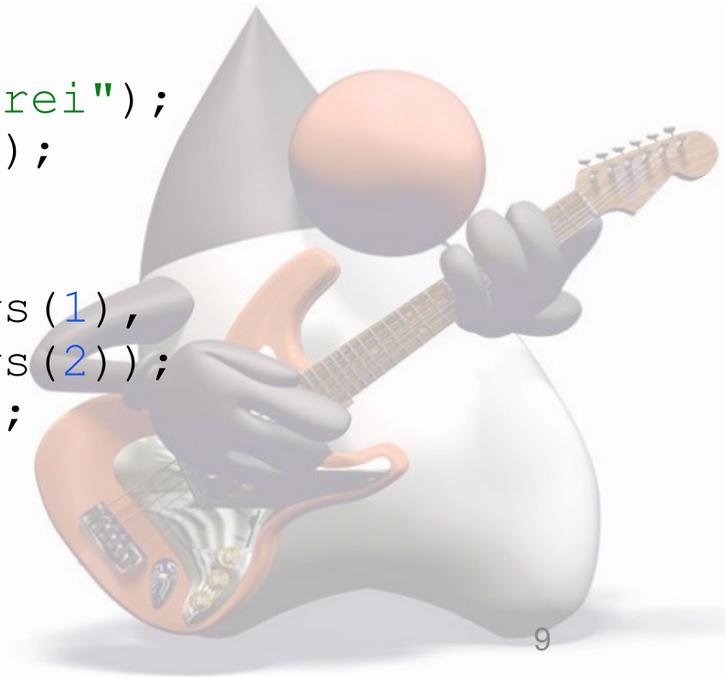
```
public static void main(String[] args) {
    NumberContainer<Integer> intContainer = new  NumberContainer<>(Integer.valueOf(4));

    //Compile error String is not a Number
    //NumberContainer<String> stringContainer = new NumberContainer<String>();
 }
}
```



# Generics

Neben typisierten Klassen gibt es auch typisierte Methoden

```
public class GenericMethod {
    public static <T> Optional<T> getLastElement(List<T> aList) {
        if (aList.isEmpty()) return Optional.empty();
        else return Optional.of(aList.get(aList.size() - 1));
    }

    public static void main(String[] args) {
        List<String> myList = List.of("Eins", "Zwei", "Drei");
        String lastElement = getLastElement(myList).get();

        List<LocalDate> dates = List.of(LocalDate.now(),
                                        LocalDate.now().plusDays(1),
                                        LocalDate.now().plusDays(2));
        LocalDate lastDate = getLastElement(dates).get();
    }
}
```



# Java Fundamentals

## Generics (Generische Klassen und Typparameter)