Java Advanced

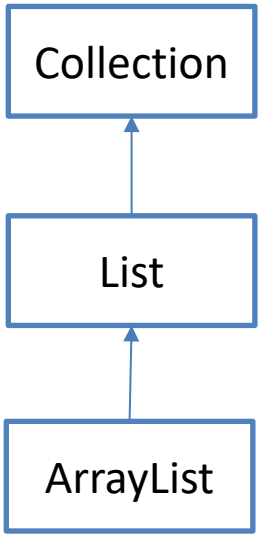
Generics / ArrayList

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

ArrayList

- Implementatie van List uit de Collections package
- Gebaseerd op array die zich dynamisch aanpast



ArrayList – generic class

Class ArrayList<E>

```
// Declare ArrayList
List<String> list = new ArrayList<>();
list.add("First");
list.add("Second");

// get element from ArrayList
list.get(0); // First
list.get(1); // Second

// loop over ArrayList
for(String item : list) {
    System.out.println(item);
}
```

ArrayList – Array

Class ArrayList<E>

```
// Decalre array of String
String[] array = { "First", "Second"};

// Convert to List of Strings
List<String> list = Arrays.asList(array);

// Convert back to array
String[] array2 = list.toArray(new String[list.size()]);
```

ArrayList - autoboxing

```
// ArrayList autoboxing

List<Integer> list = new ArrayList<>();
list.add(1); // Integer.valueOf(1);
list.add(2); // Integer.valueOf(2);
```

Generics / Inleiding

```
public class Holder {
  private Integer content;

  public Integer getContent() {
   return content;
  }
}
```

Generics / Inleiding

```
public class Holder {
  private String content;

  public String getContent() {
   return content;
  }
}
```

Generics / Inleiding

```
public class Holder {
  private Person content;

  public Person getContent() {
   return content;
  }
}
```

Generics / Inleiding

```
public class Holder {
  private Object content;

  public Object getContent() {
   return content;
  }
}
```

Generics / Inleiding

```
Holder holder1 = new Holder(1);
Holder holder2 = new Holder("Test");
Holder holder3 = new Holder(new Person(”Jos"));
…
Integer content1 = (Integer) holder1.getContent();
String  content2 = (String)  holder2.getContent();
Persoon content3 = (Persoon) holder3.getContent();
```

Generics / Werking

```
Holder<Integer> holder1 = new Holder<>(1);
Holder<String>  holder2 = new Holder<>("Test");
Holder<Person> holder3 = new Holder<>(new Person(”Jos"));
…
Integer content1 = holder1.getContent();
String  content2 = holder2.getContent();
Persoon content3 = holder3.getContent();
```

Generics / Werking

```
public class Holder<T> {
  private T content;

  public T getContent() {
   return content;
  }
}
```

Generics / Werking

```
public class Holder<T> {
  private T content;

  public T getContent() {
   return content;
  }
}

Holder<Person> holder3;
```

Generics / Gebruik

```
public class Holder<T> {
  private T content;

  public T getContent() {
   T result = content;
   return result;
  }
  public Holder(T content) {
   this.content = content;
  }
}
```

Generics / Gebruik

```
public class Holder<T> {
  private T content;

  public List<T> getAsList() {
   List<T> list = new ArrayList<>(1);
   list.add(content);
   return list;
  }
}
```

Generics / Gebruik

```
public class TextHolder extends Holder<String> {
    public TextHolder(String text) {
        super(text);
    }
}
```

Generics / Gebruik

- Andere:
  - T[] reeks
  - T mijnItem
  - for (T item : list) { … }
  - (T) obj
  - extends SuperKlasse<T>
  - …

Generics / Conventies

- Naamgeving:
  - T – Type
  - U,V, … – 2de, 3de, 4de type
  - E – Element
  - K – Key
  - N – Number
  - V – Value
  - R – Result

Generics / Werking

- Meerdere type parameters:
  - public class Vuilbak<T, U, V> { … }
  - Vuilbak<Plastiek, Metaal, Drankkarton> pmd;

Geneste generics:

- List<Houder<String>>
- Holder<List<String>>

Generics / Soorten

Type parameter

- Generische klasse
- Generische methode
- Generische interface
- Generische constructor

Generics / Methode

```
public class CoinTosser {
    private static Random rand = new Random();

    public static <T> T toss(T item1, T item2) {
        return rand.nextBoolean() ? item1 : item2;
    }
}
```

- Student jan = …, piet = …
- Student vrijwilliger = CoinTosser.toss(jan, piet);

Generics / Methode

```
public class CoinTosser {
    private static Random rand = new Random();

    public static <T> T toss(T item1, T item2) {
        return rand.nextBoolean() ? item1 : item2;
    }
}
```

- Student jan = …, piet = …
- Student vrijwilliger = CoinTosser.toss(jan, piet);

Generics / Interface

```
package java.lang;

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

Generics / Interface

```
package java.lang;

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

public final class Integer extends Number
   implements Comparable<Integer> {
   public int compareTo(Integer anotherInteger) {
    …
   }
}
```

Generics / Interface

```
package java.lang;

public final class Double extends Number
   implements Comparable<Double> {
   public int compareTo(Double anotherDouble) {
    …
   }
}
```

Generics / Methode

```
static Integer grootste(Integer a, Integer b) {
    return a > b ? a : b;
}
static Double grootste(Double a, Double b) {
    return a > b ? a : b;
}
static Card grootste(Card a, Card b) {
    return a.compareTo(b) > 0 ? a : b;
}
```

Generics / Methode

```
static Integer grootste(Integer a, Integer b) {
    return a.compareTo(b) > 0 ? a : b;
}
static Double grootste(Double a, Double b) {
    return a.compareTo(b) > 0 ? a : b;
}
static Card grootste(Card a, Card b) {
    return a.compareTo(b) > 0 ? a : b;
}
```

Generics / Methode inperken van type

```
static <T> T grootste(T a, T b) {
    return a.compareTo(b) > 0 ? a : b;
}
```

```
static <T extends Comparable<T>> T grootste(T a, T b) {
    return a.compareTo(b) > 0 ? a : b;
}
```

Generics / Syntax

- <T>
- <T extends Object>
- <T extends Number>

- <T extends Comparable<T>>
- <T extends Object & Comparable<T>>
- <T extends Number & Comparable<T>>

Generics / Syntax

- <T>
- <T extends *Class*>
- <T extends *Interface*>
- <T extends *Interface1* & *Interface2*>
- <T extends *Class* & *Interface*>
- <T extends *Class* & *Interface1* & *Interface2* & …>

Generics / Wildcards

```
public static void main(String[] args) {
    List<String> strings = new ArrayList<>();
    strings.add("een");
    strings.add("twee");
    printFirst(strings);

    List<Integer> integers = new ArrayList<>();
    integers.add(1);
    integers.add(2);
    printFirst(integers);

    List<Object> objecten = new ArrayList<>();
    objecten.add(new Person("Jos"));
    objecten.add(new Person("Eddy"));
    printFirst(objecten);
}

private static void printFirst(List<Object> list) {
    System.out.println(list.get(0));
}
```

Generics / Wildcards

```
public static void main(String[] args) {
    List<String> strings = new ArrayList<>();
    strings.add("een");
    strings.add("twee");
    printFirst(strings);

    List<Integer> integers = new ArrayList<>();
    integers.add(1);
    integers.add(2);
    printFirst(integers);

    List<Object> objecten = new ArrayList<>();
    objecten.add(new Person("Jos"));
    objecten.add(new Person("Eddy"));
    printFirst(objecten);
}

private static void printFirst(List<?> list) {
    System.out.println(list.get(0));
}
```

Achter de schermen

- Generics don’t exist at runtime!
- Er bestaat slechts 1 klasse-bestand van een generieke klasse.
- Alle informatie mbt het datatype is weggehaald in de gecompileerde klasse: type erasure
- Generieke type van de klasse is dus niet toegelaten in static variabelen
- Instanceof kan niet gebruikt worden om te controleren of een object van een gegeven generieke klasse is

Generics / Erasure

Compile time	Run time
List<T>	List<Object>
Holder<? extends Number>	Houder<Object>
Number g = getallen.get(0)	Number g = (Number) getallen.get(0)
<…>	<Object>
instanceof List<Student>	instanceof List<Object>
new T()	new Object()

Achter de schermen

```
public class GenericBox<T> {
    private static T something;
    private T item;
    private T[] items;

    public GenericBox(){
        item = new T();
        items = new T[10];
    }
```