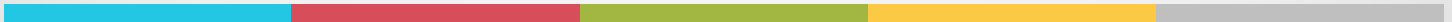
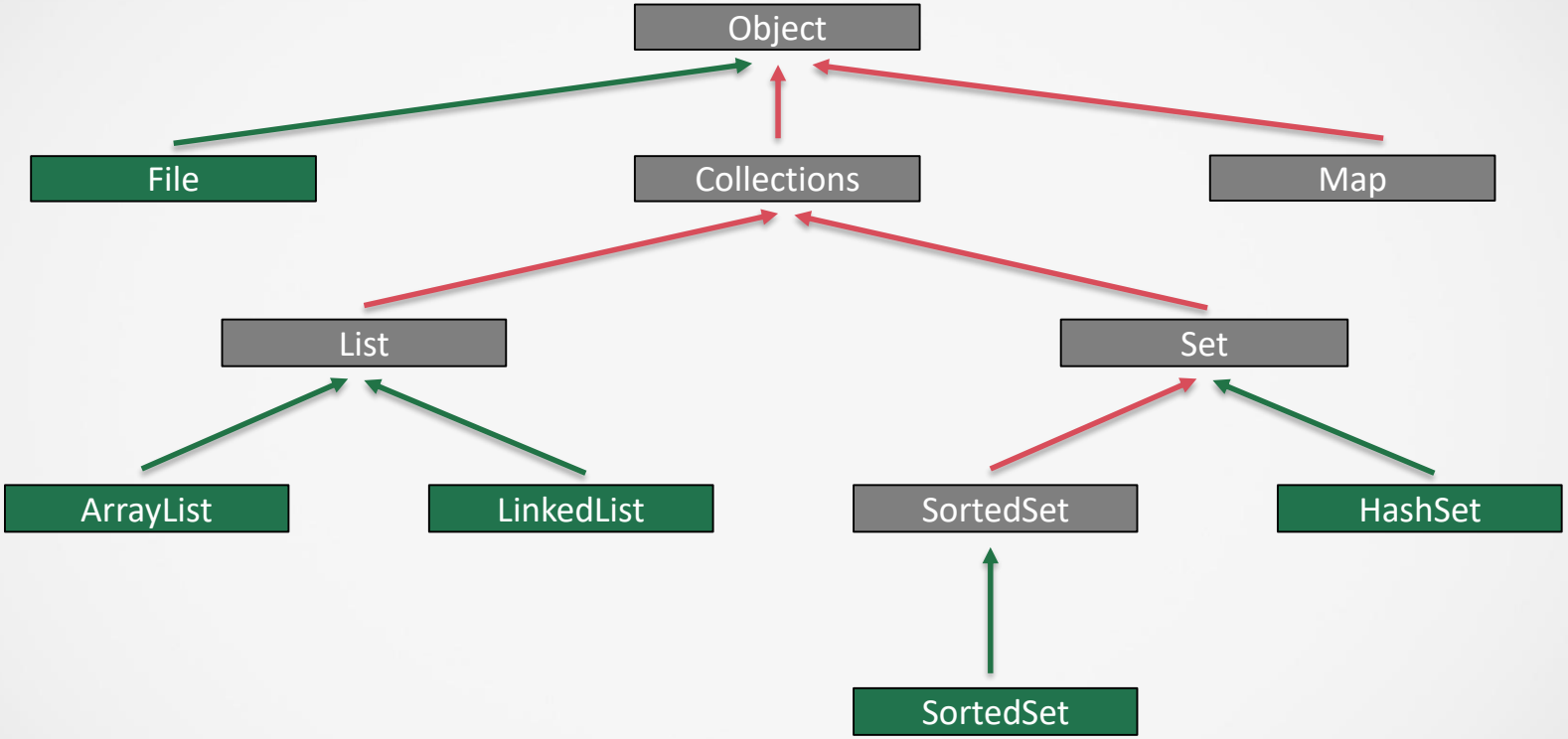
# **ADT: Java –** Collections und ArrayList



# **Überblick der** Klassen



# **ArrayList** Klasse I

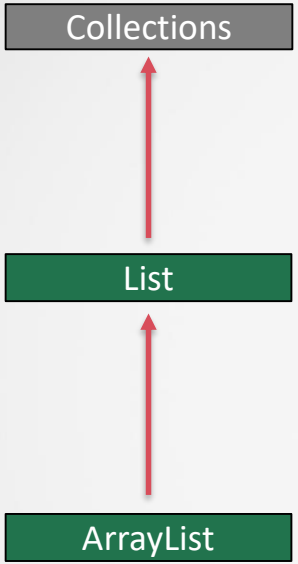
| Methode | Beschreibung |
|---|---|
| void add(int position, element obj) | Fügt ein Element an einer bestimmten Stelle in ArrayList ein. |
| boolean add(element obj) | Fügt ein Element an das Ende der ArrayList hinzu. |
| boolean addAll(Collection c) | Fügt eine Auflistung an das Ende der ArrayList hinzu. |
| element remove(int position) | Entfernt ein Element an der Position position von ArrayList. |
| boolean remove(object obj) | Entfernt das erste Vorkommen des Elements obj in der Liste ArrayList. |
| void clear() | Entfernt alle Elemente der Liste ArrayList. |
| boolean contains(Object o) | Gibt wahr zurück wenn ArrayList das entsprechende Element element enthält. |

# **ArrayList** Klasse II

| Methode | Beschreibung |
|---|---|
| object get(int position) | Gibt das Element an der Stelle position zurück |
| int indexOf(Object o) | Gibt den ersten Eintrag von dem Element o zurück oder -1 falls das Element nicht gefunden wird. |
| int lastIndexOf(Object o) | Gibt das letzte Auftreten des Elementes in der Liste zurück oder -1 wenn das Element nicht gefunden wird. |
| int size() | Gibt die Anzahl der Elemente der Liste wieder. |

Rot gefärbte Methoden werden im Abitur verwendet!
Im Abitur gibt es zu ArrayList(wird dort oft List genannt) eine Syntaxbeschreibung wie oben!
Diese kann von der Syntaxbeschreibung hier abweichen z.B. kein Rückgabewert “Boolean” usw.

# **Überblick der** Methoden in Sub/SuperKlassen




Da sich ArrayList von Collections ableitet, sind dort viele Methoden bereits implementiert:

add, remove, contains, size,…….

# **ArrayList** Beispiel I

```
import java.util.ArrayList;
public class Example1 {
   public static void main(String[] args) {
      ……
   }
}
```

Einbinden der nötigen Bibliotheken;
besser:
import java.util.*;

# ArrayList Beispiel II

```
ArrayList <String> myArrayList  = new ArrayList < String > ();
myArrayList.add("apple");
myArrayList.add("mango");
myArrayList.add("grapes");
myArrayList.add("orange");
System.out.println("Inhalt:" + myArrayList );
myArrayList .remove(2);
System.out.println("Nach dem Entfernen des Elementes Nr.2:" + myArrayList );
System.out.println("Anzahl der Elemente:" + myArrayList .size());
Iterator <String> itr = myArrayList.iterator();
while (itr.hasNext()) {
   System.out.println(itr.next());
}
```

Deklaration der Liste

Zufügen vier Elemente

Gibt die Liste aus

Entfernt den 2.ten Eintrag

Erstellt eine Iteratorvariable;
Durchläuft alle Elemente

# ArrayList Beispiel III

Ausgabe:

```
Inhalt:[apple, mango, grapes, orange]
Nach dem Entfernen des Elementes Nr.2:[apple, mango, orange]
Anzahl der Elemente:3
apple
mango
orange
```

# ArrayList Ergänzungen

## Liste löschen

```
myArrayList.clear();
```

## Liste durchlaufen – Variante A

```
for(int i=0; i < myArrayList.size(); i++)
    Object element = myArrayList.get(i);
```

## Liste durchlaufen – Variante B

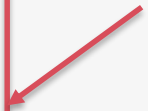
```
for(Object element: myArrayList)
  System.out.print(element);
```

# Generics

Eine Liste(und abgeleitete Klassen wie Arraylist) ist eine Liste von Zeigern auf Objekten(Instanzierungen von Klassen). Daher kann man in einer Liste jedes Objekt speichern:

```
List list = new ArrayList();
```

```
list.add(new Integer(2));
list.add("a String");
list.add(2);
list.add(new String("a String"));
```



Konstante, die Adresse wird als Zeiger gespeichert
Problem: Das Element wird als Objekt gespeichert

Beim Auslesen der Objekte müssen diese in einen anderen Objekttyp wie String oder Integer konvertiert werden;

```
Integer integer = (Integer) list.get(0);
String string   = (String) list.get(1);
```

# Generics ||

Lösung: Generics – Festlegen der Datentypen der verwendeten Instanzen

Festlegen des Datentyps des neuen Objektes

```
List<String> strings = new ArrayList<String>();

strings.add("a String");

String aString = strings.get(0);
```

Hier wird ein Wert vom Datentyp String und nicht Object hinzugefügt!

Problemloses Zuweisen des ersten Elementes an die String Variable, da get nun den Datentyp String und nicht Object zurückgibt!

Kurze Variante: `List<String> strings = new ArrayList ();`

# **Wrapper** Klassen I

Motivation: Erstellen von einem Wrapper, damit primitive Datentypen in den Java Kollektionen als Klassen verwendet werden können. Dabei wir z.B. boolean durch Boolean ersetzt:

Boolean <- boolean
Byte <- byte
Character <- char
Double <- double
Float <- float
Integer <- int
Long <- long
Short <- short

# **Wrapper** Klassen II

Erstellen oder konvertieren(3 Möglichkeiten):

```
Integer y =Integer.valueOf( "30" );
Integer y = new Integer(10);
Integer y = 10;
y++;
System.out.println("y = " + y);
```

statische valueOf

Konstruktor

“Boxing”(verwendet valueOf)

Vergleich von zwei Werten(Zwei Instanzen von Objekten):

„equals“-> siehe Strings, die Objekte sind!
Daher funktioniert dort „=“ nicht, aber natürlich equals!

# **Eigene generische** Klassen – Beispiel I

```
class GenericClass<T>{
    T t;
    public GenericClass(T t){
        this.t = t;
    }
    public void setT(T t){
        this.t = t;
    }
    public T getT(){
        return t;
    }
}
```

Allgemeiner Datentyp, kann bel. Zeichenkette sein!
Auch mehrere mit <T1, T2> möglich!

Definition des Attributes

Konstruktor

Setter und Getter des Attributes

# **Eigene generische** Klassen – Beispiel I

Lange Form

Verwendung:

```
//GenericClass<String> myGenClass = new GenericClass<String>("It must be string");
GenericClass<String> myGenClass = new GenericClass("It must be string");
 myGenClass.setT("Value Changed");
 String s = myGenClass.getT();
  //gen1.setT(new Integer(123));
}
```

Deklaration einer Variablen
Verwendung von Setter und Getter

Würde Fehler ergeben, da kein String!

# **Eigene generische** Klassen – Beispiel II

Eigene Klasse:

```
class A{
  int i;
  public A(int i) {
    this.i = i;
  }
}
```

Deklaration mit neuer Instanz von A

Verwendung:

```
GenericClass<A> gen1 = new GenericClass(new A(10));
System.out.println(gen1.getT().i);
```

getT gibt das gespeicherterte T-Objekt vom Typ A zurück, welches das (öffentliche) Attribut i hat!

## **Eigene generische** Methoden – Beispiel

```
public class TestGenericMethod {
   public static < T > void printArray( T[] inputArray ) {
      for(T element : inputArray)
         System.out.print( element+" ");
   }
 public static void main(String args[]) {
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Character[] charArray = { 'M', 'r', '.', 'B', 'i', 'g' };
      System.out.println("Array integerArray contains:");
      printArray(intArray);   // Integer array als Argument
      System.out.println("\nArray characterArray contains:");
      printArray(charArray);   // Character array als Argumet
   }
}
```

Definition & Verwendung des Objekttypen

Aufruf mit bel. Objekttypen möglich!

Literatur: http://tutorials.jenkov.com/java-generics/index.html