EXPLORE DESIGN PERFECTION

Introduction to Java

Carlos Kavka

ESTECO SpA

esteco.com

EXPLORE DESIGN PERFECTION

Introduction to Java

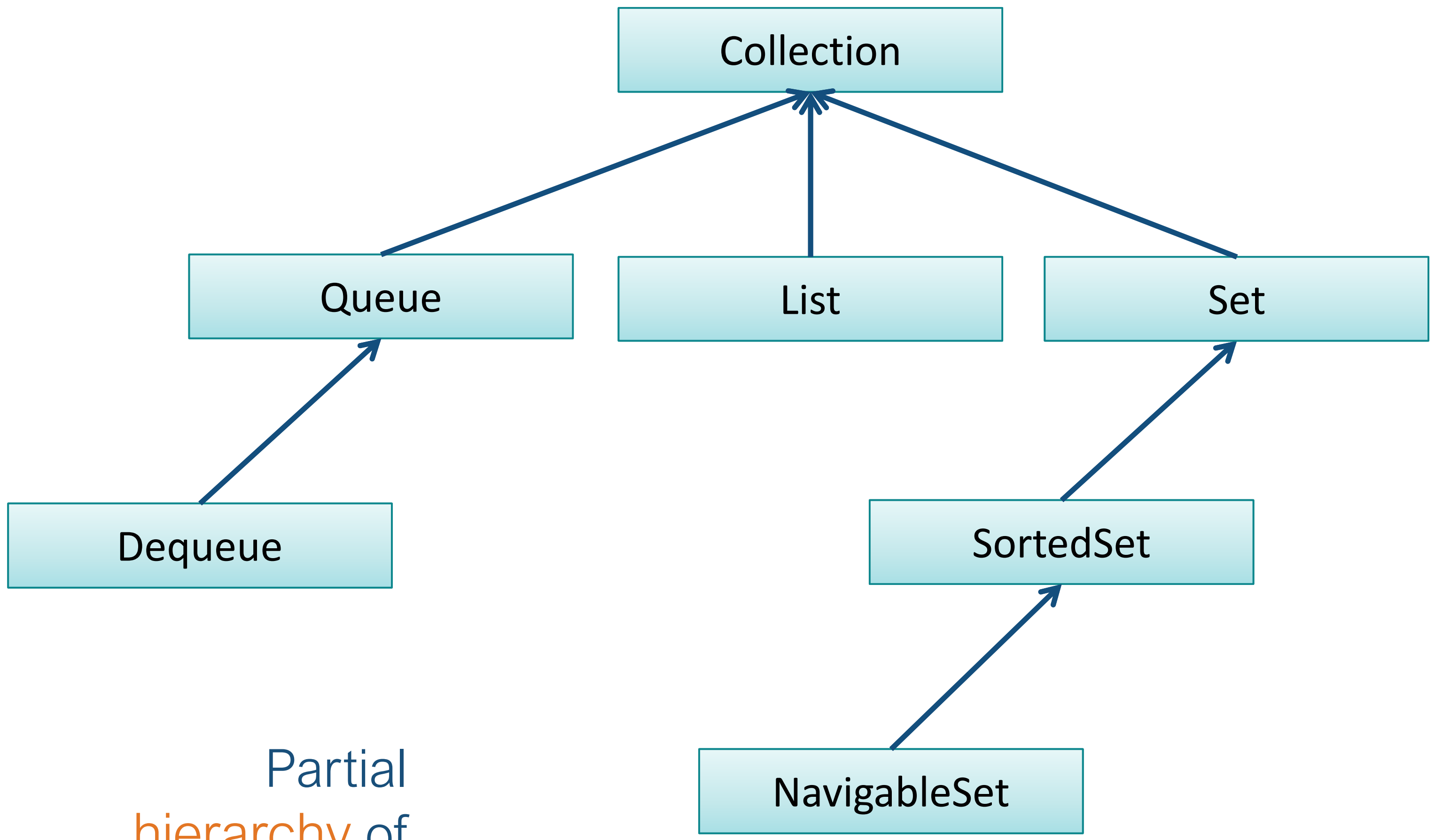
Part VII – Collections and Generics

esteco.com

Collections Framework

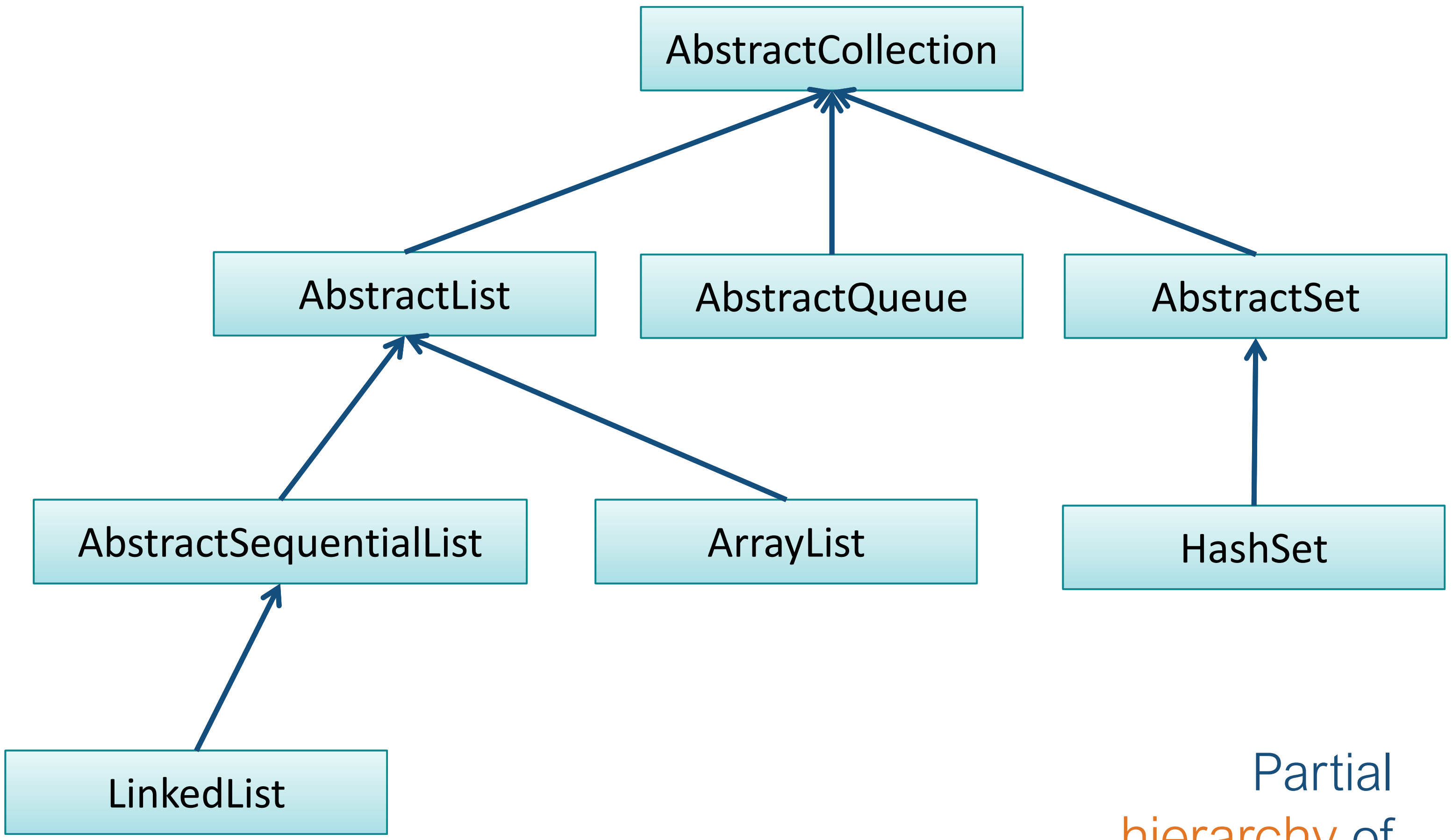
- ✓ The framework provides state-of-the-art technology for managing groups of objects
- ✓ A highly sophisticated hierarchy of interfaces and classes
- ✓ Java programmers must know and use it

Interfaces



Partial hierarchy of interfaces

Classes



Partial
hierarchy of
classes

An example with ArrayList

Creation and insertion

```
ArrayList<String> list = new ArrayList<String>();

list.add("red");
list.add("blue");
list.add("white");
```

```
for(String x : list) {
   System.out.println(x);
}
```

Traversing the structure

Removing elements

```
list.remove(2);
list.remove("white");
```

An example with LinkedList

Creation and insertion

```
LinkedList<String> list = new LinkedList<>();

list.add("red");
list.addFirst("blue");
list.add(1,"white");
```

```
for(String x : list) {
    System.out.println(x);
}
```

Traversing the structure

Removing elements

```
list.last();
list.remove("white");
```

An example with HashMap

Creation and insertion

```
 HashMap<String, Integer> map = new HashMap<>();

map.put("temperature", 22);
map.put("humidity", 65);
```

```
int temp = map.get("temperature");
```

Accessing a value

Getting keys

```
for(String x : map.keySet()) {
    System.out.println(map.get(x));
}
```

Generics

✓ Generics allows to build parameterized types:

✓ create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter.

✓ Improve type safety when compared with Objects

An example

```
public class Stack<E> {
   private LinkedList<E> data;

   Stack() {
      data = new LinkedList<E>();
   }

   public void push(E x) {
      data.addFirst(E);
   }

   public E pop() {
      return data.removeFirst();
   }

 public int size() {
      return data.size();
   }
}
```

A generic stack

```
   public static void main(String[] args) {
      Stack<Integer> stack = new Stack<>();
      stack.push(22);
      stack.push(66);
      System.out.println(stack.pop());
   }
}
```

Be careful, no checking is done when removing elements!

Bounded classes

The generic class can be restricted

```
public class Stack<BaseType extends Number> {
        …
}
```

This specifies that BaseType can only be replaced by Number, or subclasses of Number.

Wildcard arguments

Let’s defined a new methods to compare the size of two stacks:

```
public class Stack<BaseType extends Number> {
…
   public boolean equalSize(Stack<BaseType> other) {
      return size() == other.size();
   }
```

```
Stack<Integer> stack1 = new Stack<>();
stack1.push(22);
stack1.push(66);

Stack<Float> stack2 = new Stack<>();
stack2.push(3.1F);

boolean equalSize = stack1.equalSize(stack2);
```

However, it does not work if types are different!

Wildcard arguments

A new method with wildcards to compare the size of two stacks:

```
public class Stack<BaseType extends Number> {
…
   public boolean equalSize(Stack<?> other) {
      return size() == other.size();
   }
```

```
Stack<Integer> stack1 = new Stack<>();
stack1.push(22);
stack1.push(66);

Stack<Float> stack2 = new Stack<>();
stack2.push(3.1);

boolean equalSize = stack1.equalSize(stack2);
```

It works now

Comparator interface for Collections

Classes that implements the comparable interface can be "compared" by Collection methods

```
public interface Comparable<T extends Object> {

    public int compareTo(T t);
}
```

Note the bounded generic declaration!

Comparator interface for Collections

```
class Book implements Comparable<Book> {


   …


   public int compareTo(Book aBook) {
       return numberOfPages - aBook.numberOfPages;
    }
```

Books can be compared now!

```
Book b1 = new Book ("Java 8 Lambdas","Richard Warburton",168);
Book b2 = new Book("Java in a nutshell", "David Flanagan",353);

ArrayList<Book> list = new ArrayList<Book>();

list.add(b1);    list.add(b2);

Collections.sort(list);

for (Book x : list) {
  System.out.println(x.title);
}
```

Thank you for your attention!

ESTECO

EXPLORE DESIGN PERFECTION

esteco.com