

Iterators and Comparators



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

sli.do

#java-advanced

Table of Contents

1. Variable Arguments

2. Iterators

- **Iterator**
- **ListIterator**

3. Comparators

- **Comparable**





args...

Variable Arguments

Variable Arguments (Varargs)

- Allows the method to accept **zero** or **multiple** arguments

Ellipsis syntax

```
static void display(String... values) {  
    System.out.println("display method invoked");  
}  
  
static void main() {  
    display();  
    display("first");  
    display("multiple", "Strings");  
}
```

Variable Arguments Rules

- There can be **only one** variable argument **in the method**
- The variable argument **must** be the **last argument**

```
static void display(int num, String... values) {  
    System.out.println("display method invoked");  
}
```

```
void method(String... a, int... b){} //Compile time error
```

```
void method(int... a, String b){} //Compile time error
```

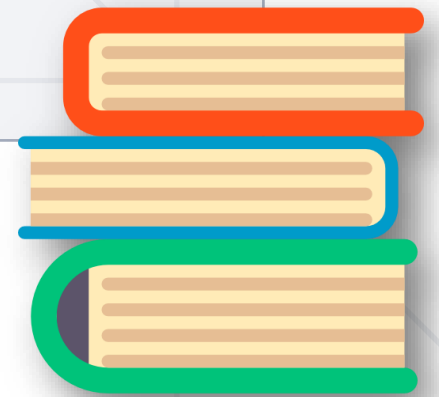
Problem: Book

- Create a class Book, which has:
 - Title
 - Year
 - Authors
- Use **only one constructor** for Book
- There can be **no authors, one author** or **many authors**

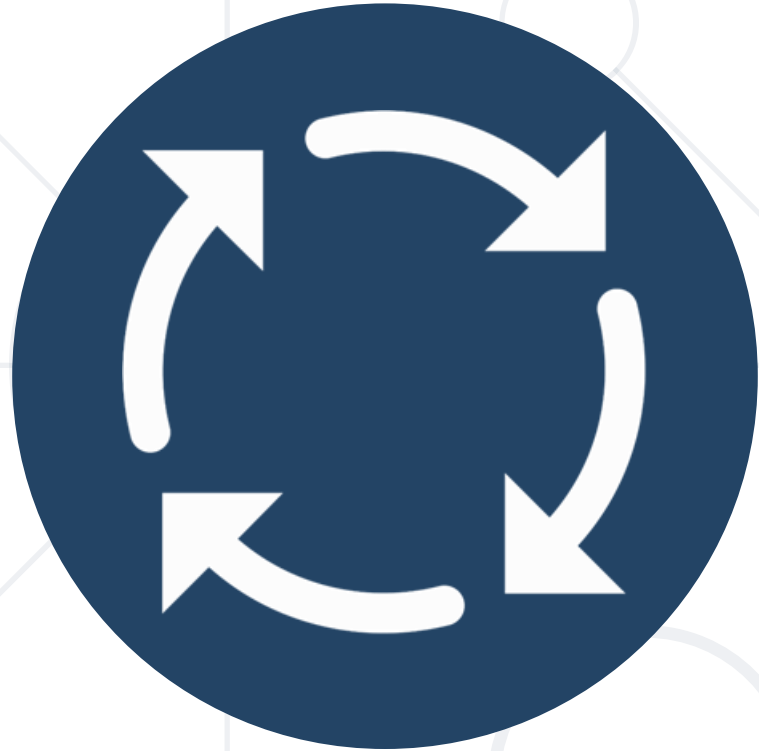


//TODO: Add fields

```
public Book(String title, int year, String... authors) {  
    this.setTitle(title);  
    this.setYear(year);  
    this.setAuthors(authors);  
}
```

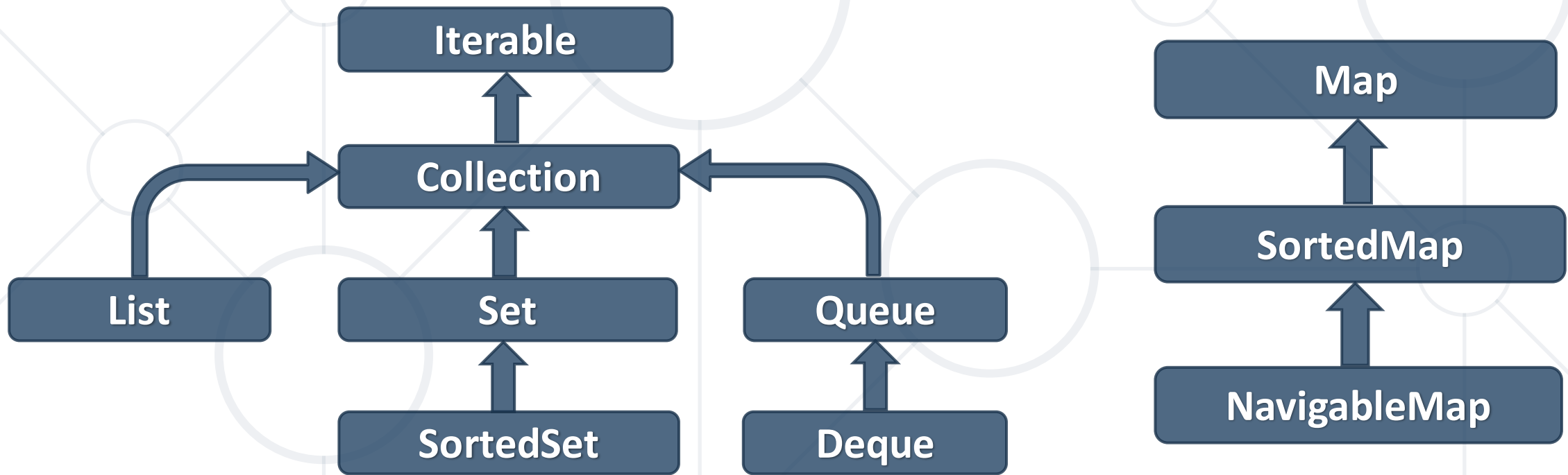



```
//TODO: Add all other getters and setters  
private void setAuthors(String... authors) {  
    if (authors.length == 0) {  
        this.authors = new ArrayList<String>();  
    } else {  
        this.authors = new ArrayList<>(Arrays.asList(authors));  
    }  
}
```




Iterable<T> and Iterator<T>

- An **Inheritance** leads to **hierarchies** of classes and/or interfaces in an application:



Iterable<T>

- Root interface of the Java collection classes
- A class that implements the **Iterable<T>** can be used with the new **for loop**



```
List list = new ArrayList();  
for(Object o : list) {  
    // do something o;  
}
```

- Abstract methods

- **iterator()**

```
public interface Iterable<T> {  
    public Iterator<T> iterator();  
}
```

- Default methods

- **forEach(Consumer<? super T> action)**

- **splititerator()** - used for parallel programming

Iterator<T>

- Enables you to cycle through a collection
- Nested class for **Iterator<T>**

```
public class Library<T> implements Iterable<T> {  
    private final class LibIterator implements Iterator<T> {}  
}
```



- Don't implement both **Iterable<T>** and **Iterator<T>**

```
class MyClass implements Iterable<T>, Iterator<T> {}
```



Problem: Library

- Create a class Library, which implements **Iterable<Book>**
- Create nested class LibIterator, which implements **Iterator<Book>**

```
<<Iterable<Book>>>  
Library
```

```
-books: Book[]
```

```
+iterator(): Iterator<Book>
```

```
<<Iterator<Book>>>  
LibIterator
```

```
-counter: int
```

```
+hasNext(): Boolean
```

```
+next(): Book
```

```
public class Library<Book> implements Iterable<Book> {  
    private Book[] books;  
    public Library(Book... books) {  
        this.books = books;  
    }  
    public Iterator<Book> iterator() {  
        return new LibIterator();  
    }  
    //TODO: Add nested iterator, look for it on next slide  
}
```



```
private final class LibIterator implements Iterator<Book> {  
    private int counter = 0;  
    public boolean hasNext() {  
        if(this.counter < books.length) { return true; }  
        return false;  
    }  
    public Book next() { return books[counter++]; }  
}
```



Comparable<T> and Comparator<T>

Comparator<E>

- The comparator provides a way for you to **provide custom comparison logic** for types that you have no control over
 - **Multiple** sorting sequence
 - **Doesn't affect** the original class
 - **compare()** method



Comparable<E>

- Comparable allows you to specify how objects **that you are implementing** get compared
 - **Single** sorting sequence
 - **Affects** the original class
 - **compareTo()** method



- Allows you to specify how objects that **you are implementing** get compared – the student's grades **st** and the **otherStudent**

```
class Student implements Comparable<Student> {
```

```
    // same as before
```

```
    @Override
```

```
    public int compareTo(Student st) {
```

```
        return Integer.compare(st.getGrades(),  
                                otherStudent.getGrades());
```

```
    }
```

```
}
```

Provide data type of
compared object

- Allows you to provide **custom comparison logic**. Compares the grades of a **st** with the grades of a **st1**:

```
class StudentGradesComparator implements Comparator<Student> {  
    // same as before  
    @Override  
    public int compare(Student st, Student st1) {  
        return Integer.compare(st.getGrades(), st1.getGrades());  
    }  
}
```

Problem: Comparable Book

- Expand Book by implementing **Comparable<Book>**
- Book has to be **compared by title**
 - When title is equal, **compare** them by **year**

```
<<Comparable<Book>>> Book

-title: String
-year: int
-authors: List<String>

-setTitle(String)
-setYear(String)
-setAuthors(String...)
+getTitle(): String
+getYear(): int
+getAuthors(): +List<String>
+compareTo(Book): int
```

Solution: Comparable Book

```
public int compareTo(Book book) {  
    if (this.getTitle().compareTo(book.getTitle()) == 0) {  
        if (this.getYear() > book.getYear()) { return 1;}  
        else if (this.getYear() < book.getYear()) { return -1; }  
        return 0;  
    } else {  
        return this.getTitle().compareTo(book.getTitle());  
    }  
}
```


Problem: Book Comparator

- Create a class, which can **compare** two books
- Use your **BookComparator** to sort list of Books

```
<<Comparator<Book>>>  
BookComparator  
+compare(Book, Book):int
```

Solution: Book Comparator

```
public class BookComparator implements Comparator<Book> {  
    @Override  
    public int compare(Book first, Book second) {  
        if (first.getTitle().compareTo(second.getTitle()) == 0) {  
            if (first.getYear() > second.getYear()) { return 1; }  
  
            // Continues on the next slide  
        }  
    }  
}
```

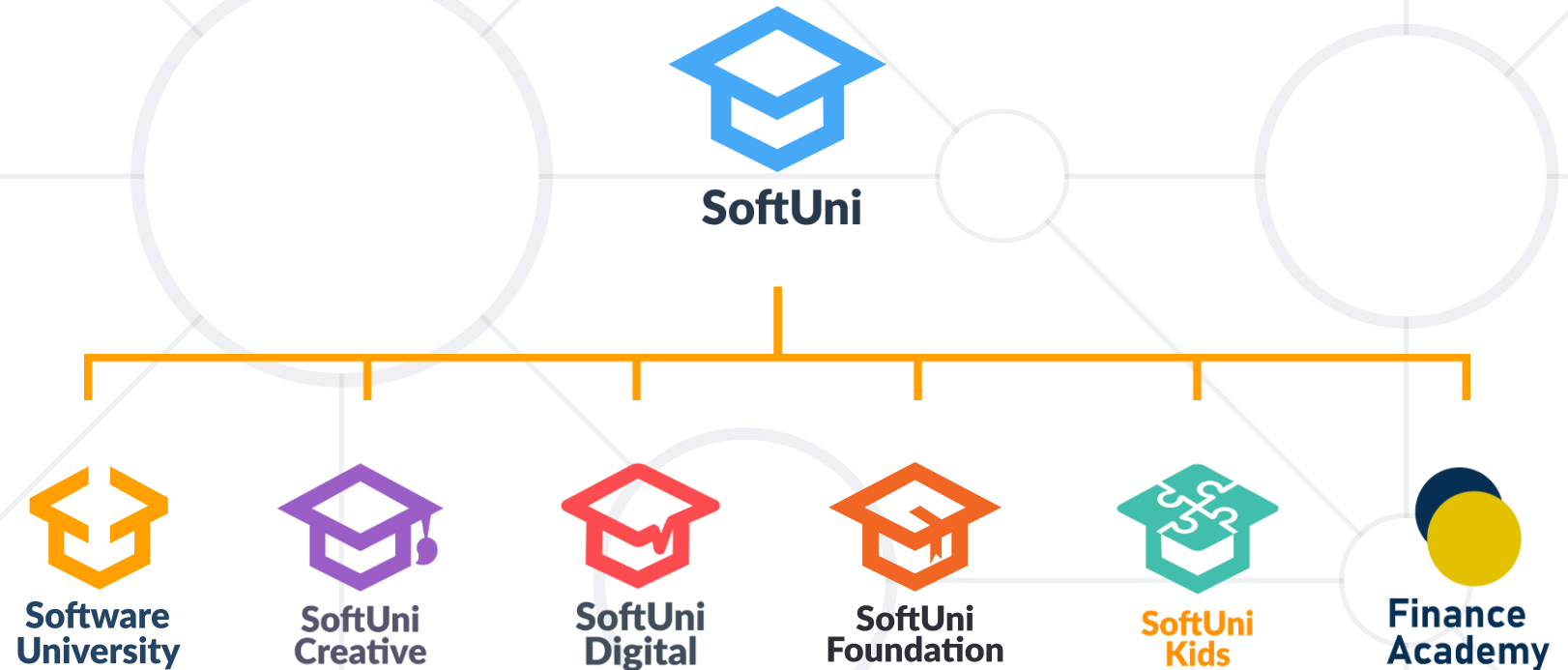
Solution: Book Comparator

```
// ...  
else if (first.getYear() < second.getYear())  
    return -1;  
return 0;  
} else {  
    return first.getTitle().compareTo(second.getTitle());  
}  
}  
}
```

- Variable arguments
- Iterable<T>
- Iterator<T>
- Comparable<T>
- Comparator<T>



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

