Introduction to

A Tiny Book Library in Java (I)


```
public class Book {
    private String author, title;

    public Book(String author, String title) {
        this.author = author; this.title = title;
    }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String toString() {
        return "Book(" + author +"," + title+ ")";
    }
}
```

A Tiny Book Library in Java (II)


```
public class JavaLibrary {

    private final List<Book> shelf = new LinkedList<>();

    public void addBook(Book book) {
        shelf.add(book);
    }

    public List<Book> booksByAuthor(String author) {
        List<Book> result = new LinkedList<>();
        for (Book b : shelf) {
            if (b.getAuthor().equals(author))
                result.add(b);
        }
        return result;
    }
    ...
```

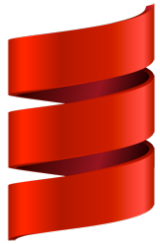
A Tiny Book Library in Java (III)


```
    ...
    public static void main(String[] args) {
      JavaLibrary lib = new JavaLibrary();
      lib.addBook(new Book("A", "A1")); lib.addBook(new Book("A", "A2"));
      lib.addBook(new Book("B", "B1"));

      List<Book> booksByA = lib.booksByAuthor("A");
      Collections.sort(booksByA, new Comparator<Book>() {
        public int compare(Book b1, Book b2) {
          return b1.getTitle().compareTo(b2.getTitle());
        }
      });
      System.out.print(booksByA);
   }
}
```

Java Pros & Cons

- **Pros**
  - Popularity and acceptance
  - Libraries, Tools
  - Java Virtual Machine (JVM)
    - Platform independent / Highly optimized: JIT, GC
  - Several Languages were defined on top of the JVM:
    - Clojure / Groovy / JRuby / Scala
- **Cons**
  - Very imperative: **How** instead of **What**
  - Not designed for highly concurrent programs
    - Original Thread model was wrong
    - It is almost impossible to write thread-safe programs
  - Verbose, too much boilerplate code (=> Eclipse: Source code generators)
    - Getters & Setters, Constructors, Equals + HashCode, …



Same in Scala

```
object ScalaLibrary {

  case class Book(author: String, title: String)

  val books = List(Book("A", "A2"), Book("A","A1"),
                   Book("B","B1"))

  def booksByAuthor(author: String): List[Book] =
    books.filter(b => b.author == author)

  def main(args: Array[String]): Unit =
    println(booksByAuthor("A").sortBy(b => b.title))
}
```

Same in Java 14 preview


```
record Book(String author, String title) {}

public class Java14Library {
  public static final List<Book> books =
    List.of(new Book("A", "A1"), new Book("A", "A2"), new Book("B", "B1"));

  public static List<Book> booksByAuthor(String author) {
    return books.stream()
            .filter(b -> b.getAuthor().equals(author))
            .collect(Collectors.toList());
  }

  public static void main(String[] args) {
    var booksByA = booksByAuthor("A").stream()
                .sorted(Comparator.comparing(b -> b.author()))
                .collect(Collectors.toList());
    System.out.print(booksByA);
  }
}
```

Scala Advantages over Java

- **In Scala functions are first class and can be passed around**
  - This encourages functional programming with all its advantages
- **In Scala immutability is the default**
  - `val elems = List(1,2,3)`
  - elems is a final reference and the referenced List is immutable
- **In Scala all values are objects (pure object-oriented)**
  - The compiler turns them into primitives, so no efficiency is lost
- **In Scala operators are just methods**
  - `a * b ⇔ a.*(b)`
- **Scala is statically typed (as Java) but uses type inference**
  - `val m = HashMap[String,List[String]]()`

Why Scala for Concurrent Programming

- **Helps to reduce code which deals with shared mutable state**
  - by promoting programming with immutable values and functions
    - Expressions yield a result instead of messing with mutable variables
- **Provides libraries which simplify concurrent programming**
  - Immutable data structures avoid race conditions
  - Software Transactional Memory Library
  - Reactive programming libraries (asynchrony without "callback hell")
  - Actor Library for coarse-grained concurrency

"If I were to pick a language to use today other than Java, it would be Scala."

-- James Gosling, creator of Java

Source: http://jonasboner.com/2009/01/30/slides-pragmatic-real-world-scala.html

I can honestly say if someone had shown me the Programming in **Scala** book by Martin Odersky [...] back in 2003 I'd probably have never created Groovy.

-- James Strachan, creator of Groovy

Source: http://macstrac.blogspot.ch/2009/04/scala-as-long-term-replacement-for.html

Teach Yourself!

- http://docs.scala-lang.org/
- https://docs.scala-lang.org/tour/tour-of-scala.html
- http://twitter.github.io/scala_school/
- http://www.scala-lang.org/api/current/
- https://docs.scala-lang.org/cheatsheets/index.html
- https://scala-lang.org/files/archive/spec/2.13/