Getting Started with Clojure

Trenton Computer Festival 2012

March 9, 2012

Michael P. Redlich
mike@redlich.net

My Background (1)

- Degree
  - ❑B.S. in Computer Science
  - ❑Rutgers University (go Scarlet Knights!)
- “Petrochemical Research Organization”
  - ❑Senior Research Technician (1988-1998, 2004-present)
  - ❑Systems Analyst (1998-2002)
- Ai-Logix, Inc. (now AudioCodes)
  - ❑Technical Support Engineer (2003-2004)
- Amateur Computer Group of New Jersey (ACGNJ)
  - ❑Java Users Group Leader (2001-present)
  - ❑Past-President (2010-present), President (2007-2009)
  - ❑Secretary (2006)

My Background (2)

- Publications
  - Java Boutique (http://www.javaboutique.com/)
    - Co-authored with Barry Burd
    - Design Patterns
  - http://www.redlich.net/publications/
- Presentations
  - Trenton Computer Festival (TCF) since 1998
  - TCF IT Professional Seminars since 2006
  - Emerging Technologies for the Enterprise since 2008
  - Princeton Java Users Group
  - Capital District Java Developers Network
  - New York Software Industry Association (NYSIA)

Objectives

- What is Clojure?
  - ❑How it evolved
  - ❑Some features of Java
  - ❑Basic differences between Java/C++
- Object-Oriented Programming Review
- Getting Started
  - ❑Includes first application and more “real world” application
- Java Beans
- Exception Handling
- Generics
- Java Database Connectivity (JDBC)

What is Java?

- "Java is C++ without guns, knives, and clubs."
  - ❑James Gosling
- "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, dynamic language."
  - ❑Sun Microsystems

Brief History of Java (1)

- Invented by James Gosling (with Patrick Naughton)
- 1991 – Originally name Oak
  - ❑consumer applications
  - ❑generate tight code
  - ❑not specific to any architecture
  - ❑object-oriented
- 1994 – “*7” Project Dissolved
  - ❑in the meantime…
- 1995 – Java introduced at Sun World ’95
  - ❑HotJava browser

Brief History of Java (2)

- 1996 – JDK 1.0
  - ❑shortly after release of Netscape 2.0
  - ❑applets only
- 1997 – JDK 1.1
  - ❑JavaBeans, JDBC, Reflection, RMI, AWT
- 1998 – JDK 1.2 (Java 2)
  - ❑Java Foundation Classes (JFC), consistent "look and feel"
- 2004 – JDK 1.5 (Java 5)
  - ❑Generics, enum, autoboxing, static import
- 2006 – JDK 1.6 (Java 6)
  - ❑Current release 1.6.0_24

Some Java Features

- Object-Oriented Programming language
- Automatic documentation
- Applets and applications
- Comprehensive exception handling
- Java Database Connectivity (JDBC)
- JavaBeans/Enterprise Java Beans
- No pointers!!

Basic Differences Between Java and C++ (1)

- Pointers
  - ❑none in Java
- Destructors
  - ❑none in Java
- Inheritance
  - ❑only single inheritance in Java
- Constant methods (member functions)
  - ❑none in Java

Basic Differences Between Java and C++ (2)

- Standard Template Library
  - ❑none in Java until...
  - ❑...generics first implemented in Java 5 comes very close
- Header Files
  - ❑none in Java

Object-Oriented Programming Review (1)

- A programming paradigm
  - ❑procedure-oriented
  - ❑object-oriented
- Four Main Attributes
  - ❑data encapsulation
  - ❑data abstraction
  - ❑inheritance
  - ❑polymorphism

Object-Oriented Programming Review (2)

Procedure-Oriented

- Top down
- Bottom up
- Structured programming
- Centered around an algorithm
- Identify tasks; how something is done

Object-Oriented

- Identification of objects to be modeled
- Concentrate on what an object does
- Hide how an object performs its tasks
- Identify an object’s behavior and attributes

Object-Oriented Programming Review (3)

- Abstract Data Type (ADT)
    - user-defined data type
    - use of objects through provided functions without knowing the internal representation
- Interface
    - the provided functions in the ADT that allow access to data
- Implementation
    - the underlying data structure(s) in the ADT

Object-Oriented Programming Review (4)

<u>Class</u>

- Defines a model
- Declares attributes
- Declares behavior
- An ADT

<u>Object</u>

- An instance of a class
- Has state
- Has behavior
- Many *unique* objects of the same class

Advantages of Object-Oriented Programming

- Implementation can be refined and improved without having to change the interface
- Encourages modularity in program development
- Better maintainability of code
- Code reuse
- Emphasis on *what*, not *how*

Some Java Keywords

- `class`
- `new`
- `private`
- `protected`
- `public`
- `package`
- `final`
- `try`
- `throw`
- `catch`
- `finally`
- `implements`
- `extends`
- `abstract`

Java Development Kit (JDK)

- JDK available from Oracle's web site
  - ❑ http://java.sun.com/
  - ❑ Java SE (Standard Edition)
  - ❑ Latest version: Java 6 (1.6.0) update 24
  - ❑ Available for Solaris, Linux, and Win9x/NT/2000/XP
- JDK documentation available separately
  - ❑ full HTML format

Laboratory Exercise #1

- Setup Your Java Environment

Working with Java (1)

- Source code
  - ❑File(s) with `.java` extension
- Intermediate bytecode
  - ❑Generated `.class` file(s) after successful compilation
- Bytecode interpreted by Java Virtual Machine (JVM)
- Set environment variable and path
  - ❑`set JAVA_HOME = C:\jdk1.6.0_24`
  - ❑`set PATH = %PATH%;%JAVA_HOME%\bin`

Working with Java (2)

- Compile Java source code

  ```
  C:\> javac -Xlint:all -d [path] File.java
  ```

- Invoke an application

  ```
  C:\> java -classpath [path] File
  ```

- Invoke an applet
  - In browser via HTML file containing `<applet></applet>` tags

  ```
  C:\> appletviewer file.html
  ```

Classes

- User-defined abstract data types
- Contain:
  - ❑ Constructor
  - ❑ Data members
  - ❑ Methods (member functions)
- One consistent instantiation mechanism
- Multiple constructors
  - ❑ `Sports(String team,int win,int loss)`
  - ❑ `Sports(float pct,String team,int win)`
- Abstract Class
  - ❑ Declares at least one abstract method

Class Instantiation

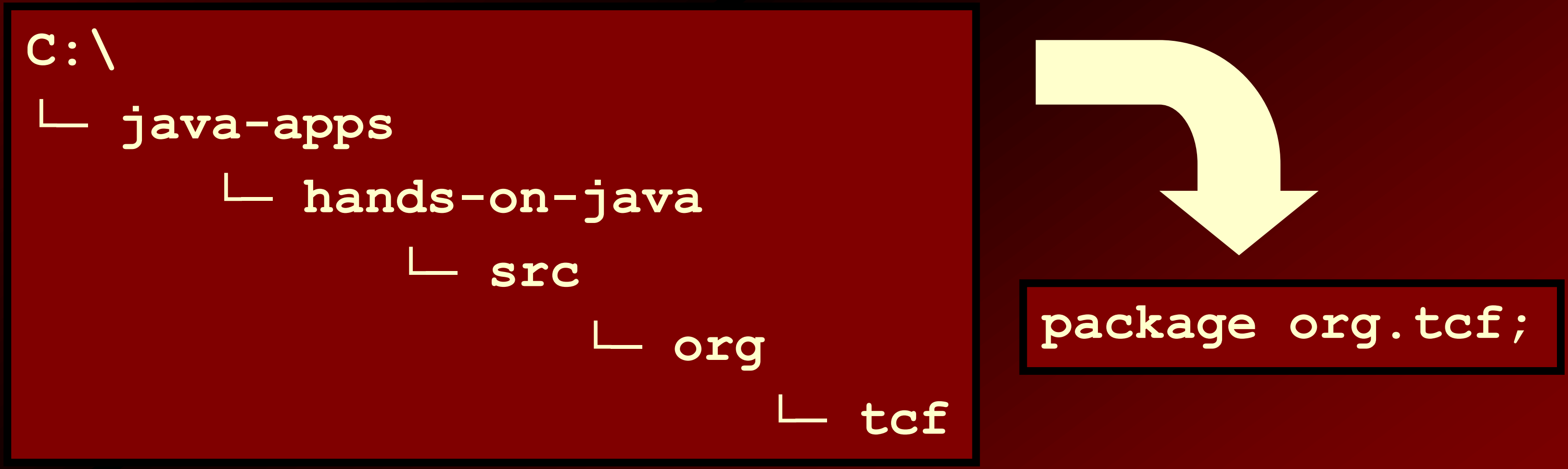
- Object creation
  - ❑`Baseball mets = new Baseball(“Mets”,97,65);`
- Access to public member functions
  - ❑`int win = mets.getWin();`
- Object deletion
  - ❑Automatic garbage collection
  - ❑`System.gc()`

Laboratory Exercise #2

- Your First Java Application

Directories & Packages

- Consistent directory structure
  - Source code (`*.java`)
  - Generated byte code (`*.class`)
- Map directories with package name under the `src` folder



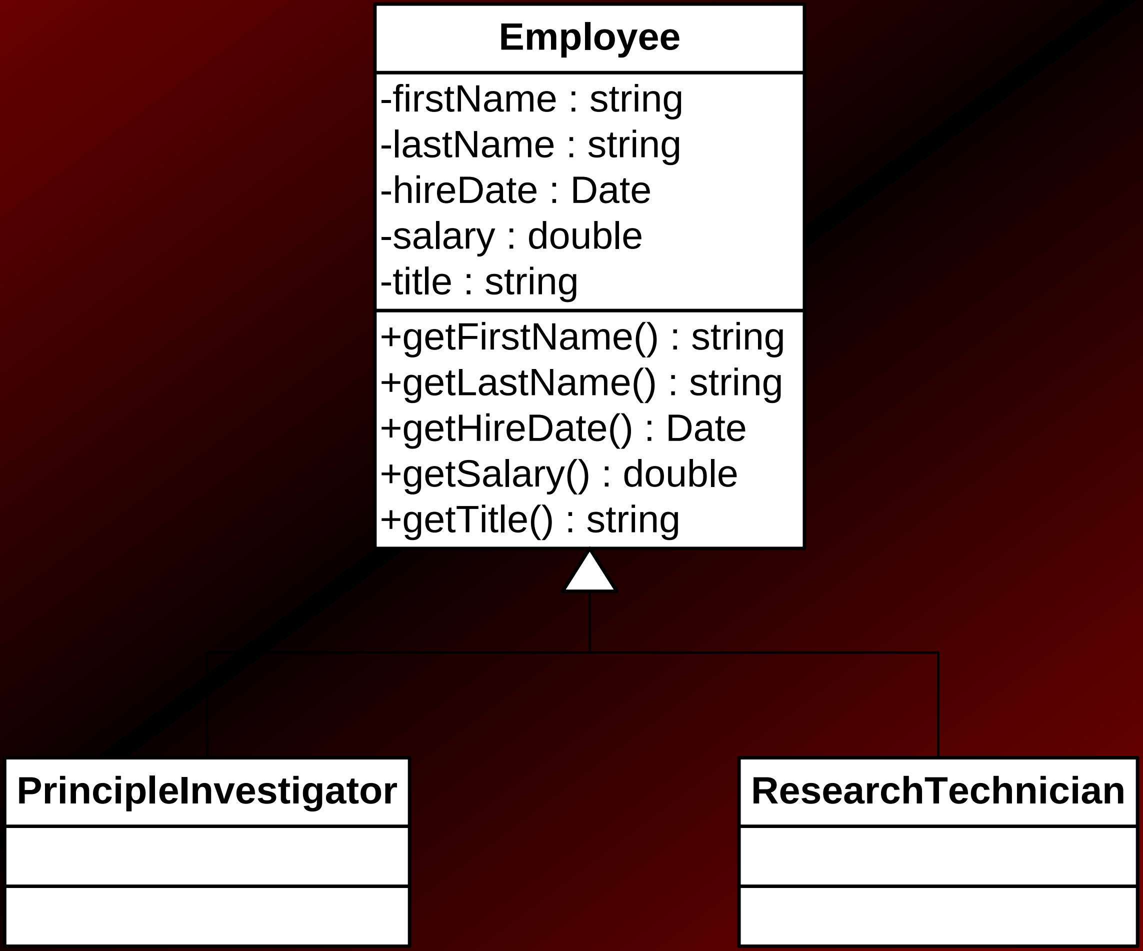
```
C:\
└─ java-apps
      └─ hands-on-java
            └─ src
                  └─ org
                        └─ tcf
```

```
package org.tcf;
```

Laboratory Exercise #3

- Establishing Directories for Packages

Class Inheritance



Laboratory Exercise #4

- A More "Real World" Java Application

Java Beans (1)

- A method for developing reusable Java components
- Also known as:
  - ❑POJOs (Plain Old Java Objects)
- Easily store and retrieve information
- A Java class is considered a bean when it:
  - ❑Implements interface `Serializable`
  - ❑Defines a default constructor
  - ❑Defines properly named getter/setter methods

Java Beans (2)

- Public Getter/Setter methods
  - Assign (set) and return (get) a bean’s data members
  - Follow specified naming convention
    - `getName/setName`
    - Where `name` is the name of the private data member
  - Follow specified boolean naming convention
    - `isValid/setValid`
    - Where `valid` is the name of the private boolean value

```
public class SportsBean implements Serializable {
    private int win;
    private boolean empty;

    public SportsBean() {
        }

    public int getWin() {
        return win;
        }

    public void setWin(int win) {
        this.win = win;
        }

// continued on next slide...
// continued from previous slide...

    public boolean isEmpty() {
        return empty;
        }

    public void setEmpty(boolean empty) {
        this.empty = empty;
        }
    }
```

Laboratory Exercise #5

- Java Beans

Exception Handling (1)

- More robust method for handling errors than fastidiously checking for error codes
  - Error code checking is tedious and obscures the program logic
- The Java Exception Model
  - Checked exceptions
    - Enforced by the compiler
  - Unchecked exceptions
    - Not enforced by compiler
  - Exception specifications
    - Specify what type of exception(s) a function will throw
  - Termination vs. resumption semantics

Exception Handling (2)

- throw-expression
  - ❑ Raises the exception
  - ❑ `throw Throwable;`
    - ❖ Where `Throwable` is an instance of a class that extends `Throwable`
- try-block
  - ❑ Contains a throw expression or a function that throws an exception

Exception Handling (3)

- catch clause(s)
  - ❑ Handles the exception
  - ❑ Defined immediately after the try-block
  - ❑ Multiple catch clauses can be defined
    - ❖ Should be ordered from most significant to least significant
  - ❑ Implicit data type conversions will not work
- finally clause
  - ❑ Always get called regardless of what happens with the exception and where it is caught
  - ❑ Set something back to its original state other than memory allocation

Exception Handling (4)

- Do not throw exceptions…
  - ❑…to indicate special return values

```
// ExceptionTest class

public class ExceptionTest {
    public static void main(String[] args) {
        try {
            initialize();
            }
        catch(Exception exception) {
            exception.printStackTrace();
            }
        }

    public void initialize() throws Exception {
        // contains code that may throw an Exception
        // type as specified
        }
    }
```

Laboratory Exercise #6

- Exception Handling (to be developed)

Generics

- A mechanism to ensure type safety in Java Collections
- Introduced in Java 5
- Similar concept to the C++ Template mechanism
    - Except no multiple copies of code
- Prototype:

```
[visibility-modifier] class | interface name<Type> {
    // body of class or interface...
    }
```

Before Generics…

```
// List example

List list = new ArrayList();
for(int i = 0;i < 10;++i)
    list.add(new Integer(i));
Iterator iterator = list.iterator();
while(iterator.hasNext())
    System.out.println(“i = ” + (Integer)iterator.next());
```

After Generics…

```
// List example

List<Integer> list = new ArrayList<Integer>();
for(int i = 0;i < 10;++i)
    list.add(new Integer(i));
Iterator iterator = list.iterator();
while(iterator.hasNext())
    System.out.println(“i = ” + iterator.next());
```

Defining Simple Generics

```
public interface List<E> {
    add(E x);
    }

public interface Iterator<E> {
    E next();
    boolean hasNext();
    }
```

Laboratory Exercise #7

- Generics (to be developed)

Java Database Connectivity (JDBC) (1)

- A built-in API to access data sources
  - ❑Relational databases
  - ❑Spreadsheets
  - ❑Flat files
- The JDK includes a JDBC-ODBC bridge for use with ODBC data sources
  - ❑Type 1 driver

Java Database Connectivity (JDBC) (2)

- Install database driver and/or ODBC driver
- Establish a connection to the database
  - ❑ Load database driver
    - ❖ `Class.forName(`*`driverName`*`);`
  - ❑ Make database connection
    - ❖ `DriverManager.getConnection();`

Java Database Connectivity (JDBC) (3)

- Create JDBC statement(s)
  - ❑Send SQL statement(s) to the database
    - ❖`connection.createStatement();`
- Obtain result set(s)
  - ❑Execute statements
    - ❖`statement.execute();`
    - ❖`statement.executeQuery();`

```
import java.sql.*;

public class DBTest {
    static public void main(String[] args) {
        String sql = “SELECT * FROM tblTimeZones”;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection connection =
            DriverManager.getConnection
            ("jdbc:odbc:timezones","","");
        Statement statement =
            connection.createStatement();
        ResultSet result =
            statement.executeQuery(sql);
        while(result.next())
            System.out.println(result.getDouble(2)
                + " " + result.getDouble(3));
        connection.close();
        }
    }
```

Laboratory Exercise #8

- Java Database Connectivity (JDBC) (to be developed)

The Java 2 Collections

- Containers before Java 2 were a disappointment
  - ❑Only four containers
  - ❑No built-in algorithms
- Java 2 collections inspired by C++'s Standard Template Library (STL)
- Two families of containers
  - ❑Collections
  - ❑Maps

Containers

- *Sequential containers* organize elements linearly
- *Sorted associative containers* organize objects based on a key for quick retrieval of data
- Primarily chosen by how well it can perform certain operations
  - ❑Add elements to the container
  - ❑Remove elements from the container
  - ❑Rearrange elements within the container
  - ❑Inspect elements within the container

Collections (1)

- Implement the `Collection` interface
- Built-in implementations:
  - ❑`List`
  - ❑`Set`

Collections (2)

- Lists
  - ❑ ordered sequences that support direct indexing and bi-directional traversal
- Sets
  - ❑ an unordered receptacle for elements that conform to the notion of a mathematical set
  - ❑ duplicates not allowed

```
// the Collection interface

public interface Collection {
    boolean add(Object object);
    boolean addAll(Collection collection);
    void clear();
    boolean contains(Object object);
    boolean containsAll(Collection collection);
    boolean equals(Object object);
    int hashCode();
    boolean isEmpty();
    Iterator iterator();
    boolean remove(Object object);
    boolean removeAll(Collection collection);
    boolean retainAll(Collection collection);
    int size();
    Object[] toArray();
    Object[] toArray(Object[] array);
    }
```

Collections (3)

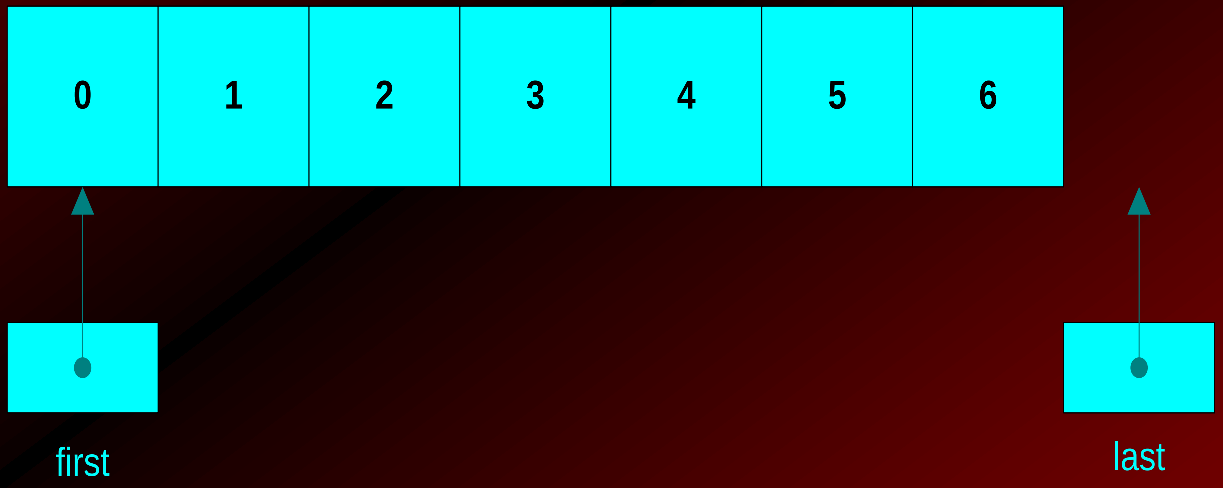
	`vector`	`deque`	`list`	`set/map`
insert/erase	$O(n)$	$O(n)$	$O(1)$	$O(n\log n)$
prepend	$O(n)$	$O(1)$	$O(1)$	$O(n\log n)$
find(val)	$O(n)$	$O(n)$	$O(n)$	$O(n\log n)$
X[n]	$O(1)$	$O(1)$	$O(n)$	$O(n)$
no. of pointers	0	1	2	3

Iterators

- Used to access elements within an ordered sequence
- All collections support iterators
- Traversal depends on the collection
- All iterators are *fail-fast*
  - ❑If data structure is changed by something other than an iterator, the iterator becomes invalid

```
import java.util.*;

List<Integer> list = new ArrayList<Integer>();
for(int i = 0;i < 7;++i)
    list.add(new Integer(i));
Iterator iterator = list.iterator();
while(iterator.hasNext())
    System.out.print(iterator.next());
```



Java IDEs (1)

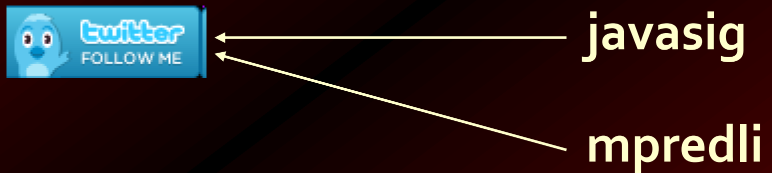
- JetBrains IntelliJ IDEA
  - ❑ http://www.jetbrains.com/idea/
- Eclipse
  - ❑ http://www.eclipse.org/
- Embarcadero JBuilder
  - ❑ http://www.embarcadero.com/products/jbuilder/
- Sun NetBeans
  - ❑ http://www.netbeans.org/

Java IDEs (2)

- Provide:
  - ❑Automatic code generation
  - ❑Context sensitive help
  - ❑Plug-ins
  - ❑Integration with Ant

Java Resources (1)

- ACGNJ Java Users Group
  - ❑Facilitated by Mike Redlich
  - ❑http://www.javasig.org/



javasig

mpredli

- Princeton Java Users Group
  - ❑Facilitated by Yakov Fain
  - ❑http://www.myflex.org/princetonjug/
- NYJavaSIG
  - ❑Facilitated by Frank Greco
  - ❑http://www.javasig.com/

twitter
FOLLOW ME

nyjavasig

Java Resources (2)

- Capital District Java Developers Network
  - ❑ facilitated by Anthony DeBonis
  - ❑ http://www.cdjdn.org/
- Sun's Java web site
  - ❑ http://java.sun.com/
- Java Boutique
  - ❑ http://www.javaboutique.com/
- Java Ranch
  - ❑ http://www.javaranch.com/

Java Resources (3)

- java.net
  - ❑ http://www.java.net/
- redlich.net
  - ❑ http://www.redlich.net/publications/
  - ❑ Slides for all TCF presentations
  - ❑ Demo Java application

Further Reading (1)

- Java 2 for Dummies, 2nd Edition
  - ❑Barry Burd
  - ❑ISBN 0-7645-6858-2
  - ❑http://www.barryburd.com/
- The Java Tutorial for the Real World
  - ❑Yakov Fain
  - ❑ISBN 0-9718439-0-2
  - ❑http://www.smartdataprocessing.com/

Further Reading (2)

- Head First Java, 2nd Edition
  - ❑ Kathy Sierra and Bert Bates
  - ❑ ISBN 0-596-00920-8
  - ❑ http://www.wickedlysmart.com/
- Thinking in Java
  - ❑ Bruce Eckel
  - ❑ ISBN 0-13-027363-5
  - ❑ http://www.bruceeckel.com/
- Java Developers Journal
  - ❑ http://java.sys-con.com/