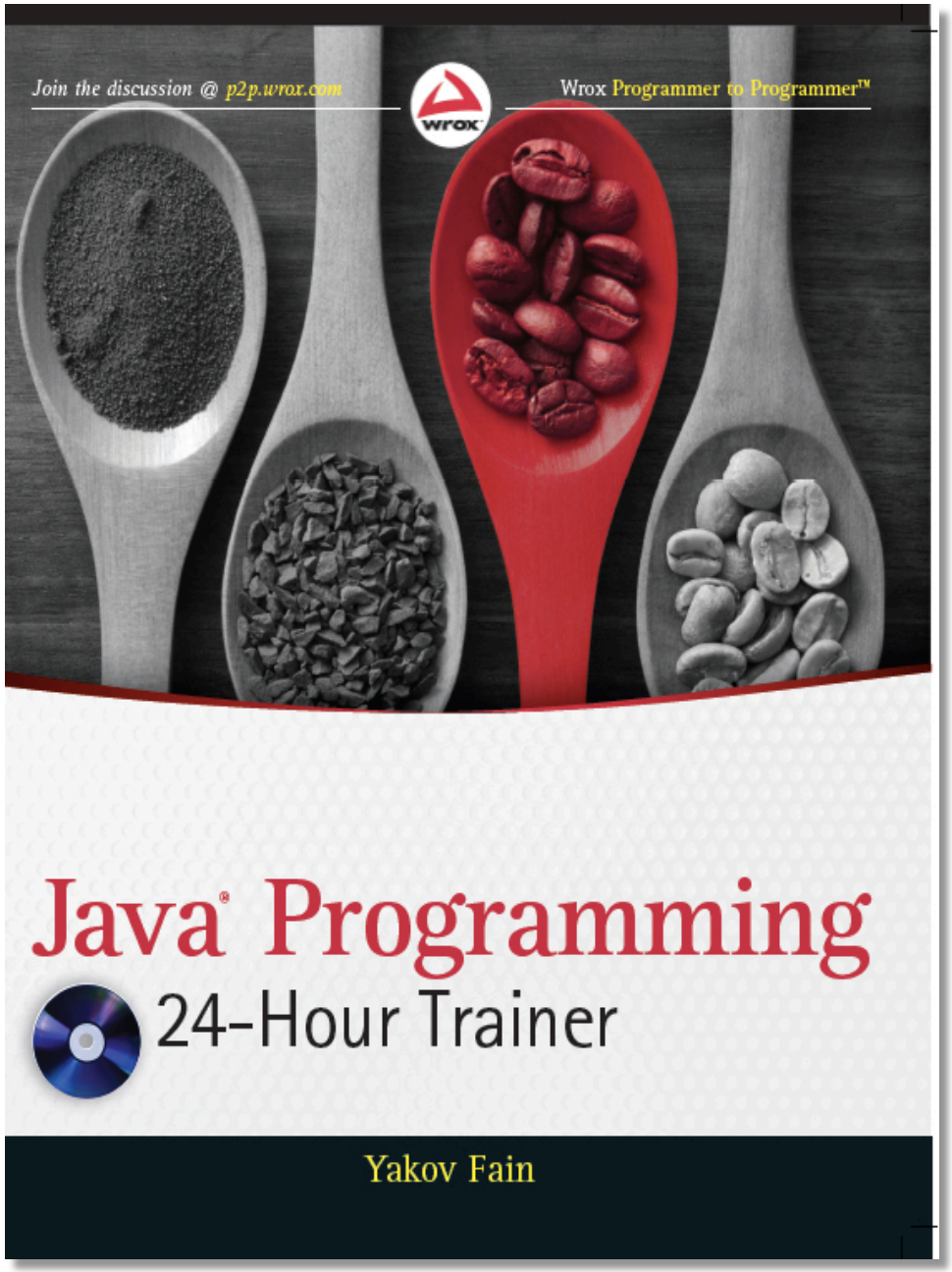
Java 101 for Flex Developers

By Yakov Fain, Farata Systems

FARATA THE EXPERT CONSULTANCY

WE BUILD APPLICATIONS. EVERY APP IS UNIQUE.
WE CREATE IT. YOU OWN IT.



I'll be using materials from my book "Java Programming 24-Hour Trainer".

JDK and JRE

- Java Development Kit (JDK) is required to develop and run programs.
- Java Runtime Environment (JRE) is required to run programs.
- Users must have JRE installed, developers – JDK.

Java SE and Java EE

- Java SE: Java Standard Edition is available at

  http://www.oracle.com/technetwork/java/**javase**/downloads

- Java EE: Java Enterprise Edition (a.k.a. J2EE)

- Java EE includes a set of technologies built on top of Java SE: Servlets, JSP, JSF, EJB, JMS, et al.

- All Java programs *run* inside the Java Virtual Machine (JVM) similarly to compiled ActionScript that runs in a VM a.k.a. Flash Player.

Running a Java program without IDE

1. Write the program and save it in a file with the name that ends with .java, for example HelloWorld.java

2. Compile the program using javac compiler, e.g. javac HelloWorld.java

   This will create a file HelloWorld.class

3. Run your program: java HelloWorld

Eclipse IDE

- Eclipse is the most widely used IDE.
- Alternatives: IntelliJ IDEA (JetBrains), NetBeans (Oracle).
- Download Eclipse IDE for Java EE developers at eclipse.org.
- You still have to download and install JDK separately.
- Flash Builder is built on top of Eclipse IDE.

Hello World Demo in Eclipse IDE

Variable and constants

In Java you don't use the keyword var. Data type goes first.

```
int currentSubmergeDepth;                // integer primitive variable
boolean isGunOnBoard=true;               //  boolean primitive variable
final String MANUFACTURER="GAZ";         //  a final String variable
```

First declare a variable, then use it.

```
currentSubmergeDepth = 25;
```

You can assign the value to a final variable **only once** and can't change it afterward.

```
MANUFACTURER = "Toyota";
```

Method Signature

In the method signature you need to declare the data type and the name of each argument:

```
int calcLoanPayment(int amount, int numberOfMonths, String state){

    // Your code goes here
    return 12345;
}
```

The method return type goes first.

You can call this method passing the values for the payment calculations as arguments:

```
calcLoanPayment(20000, 60, “NY”);
```

Java Classes

```
class Car{
   String color;
   int numberOfDoors;

   void startEngine() {
     // Some code goes here
   }
   void stopEngine () {
       int tempCounter=0;
     // Some code goes here
   }
}
```

```
class TestCar{

   public static void main(String[] args){

      Car  car1 = new Car();
      Car  car2 = new Car();
       car1.color="blue";
       car2.color="red";

       // Printing a message on the console like trace() in ActionScript
        System.out.println("The cars have been painted ");
   }
}
```

Inheritance works as in ActionScript

```
class Tax {
    double grossIncome;
    String state;
    int  dependents;


    public double calcTax() {

            return 234.55;
    }


}
```

```
class NJTax extends Tax{

   double adjustForStudents (double stateTax){
      double adjustedTax = stateTax – 500;
      return adjustedTax;
   }
}
```

Abstract Classes

A class is called abstract if it was declared with the `abstract` keyword. You can not instantiate an abstract class. Usually, an abstract class has at least one abstract method.

```
abstract public class Person {

    public void changeAddress(String address){
            System.out.println("New address is" + address);
     }
 ...
  // an abstract method to be implemented in subclasses
   public abstract boolean increasePay(int percent);
}
```

The increasePay() method must be implemented in the subclasses of Person, which may implement it differently, but the signature of the method increasePay() will be the same.

Abstract classes are not supported by ActionScript 3.

Promoting Workers. The spec.

A company has employees and contractors. Design the classes without using interfaces to represent the people who work for this company.

The classes should have the following methods:

changeAddress()
promote()
giveDayOff()
increasePay()

Promotion means giving one day off and raising the amount in the pay check.

For employees, the method increasePay() should raise the yearly salary.

For contractors, the method increasePay() should increase their hourly rate.

```
abstract public class Person {

        private String name;
        int INCREASE_CAP = 20;  // cap on pay increase

         public Person(String name){
                this.name=name;
         }

         public String getName(){
                return "Person's name is " + name;
         }

         public void changeAddress(String address){
                    System.out.println("New address is" + address);
         }

         private void giveDayOff(){
                    System.out.println("Giving a day off to " + name);
         }

      public void promote(int percent){
                    System.out.println(" Promoting a worker...");
                    giveDayOff();

                    //calling an abstract method
                    increasePay(percent);
          }
          // an abstract method to be implemented in subclasses
          public abstract boolean increasePay(int percent);
}
```

Interfaces used similarly to ActionScript

- *Interfaces* can contain only declarations of methods *and final variables*

```
public interface Payable {
    boolean increasePay(int percent);
}
```

- A class can implement one or more interfaces

```
class Employee implements Payable, Promotionable {…}

class Contractor implements Payable{…}
```

- If a class declaration has the `implements` keyword it MUST implement every method that's declared in the interface(s) that this class implements.

Casting has different syntax

All Java classes form an inheritance tree with the class `Object`. While declaring non-primitive variables you are allowed to use either the exact data type of this variable or one of its ancestor data types. For example, if the class `NJTax` extends `Tax` each of these lines is correct.

```
NJTax myTax1   = new NJTax();
Tax myTax2     = new NJTax();  // upcasting
Object myTax3  = new NJTax();  // upcasting
```

If `Employee` and `Contractor` extend class `Person`, you can declare array of type `Person`, but populate it with employees and contractors:

```
Person workers[] = new Person [100];

workers[0] = new Employee("Yakov", "Fain");
workers[1] = new Employee("Mary", "Lou");
workers[2] = new Contractor("Bill", "Shaw");
```

Casting (cont.)

Placing a data type in parenthesis in front of another type means that you want to *cast* this object to specified type.

```
Person workers[] = new Person [20];

// Code to populate the array workers with Person's descendants goes here.

for (int i=0; i<20; i++){
    Employee currentEmployee;
    Contractor currentContractor;

    if (workers[i] instanceof Employee){                    // type check

        currentEmployee = (Employee) workers[i];            // downcasting
        // do some employee-specific processing here

    } else if  (workers[i] instanceof Contractor){

       currentContractor = (Contractor) workers[i];         // downcasting
       // do some contractor-specific  processing here
    }
}
```

Demo of the Abstract classes

Polymorphism

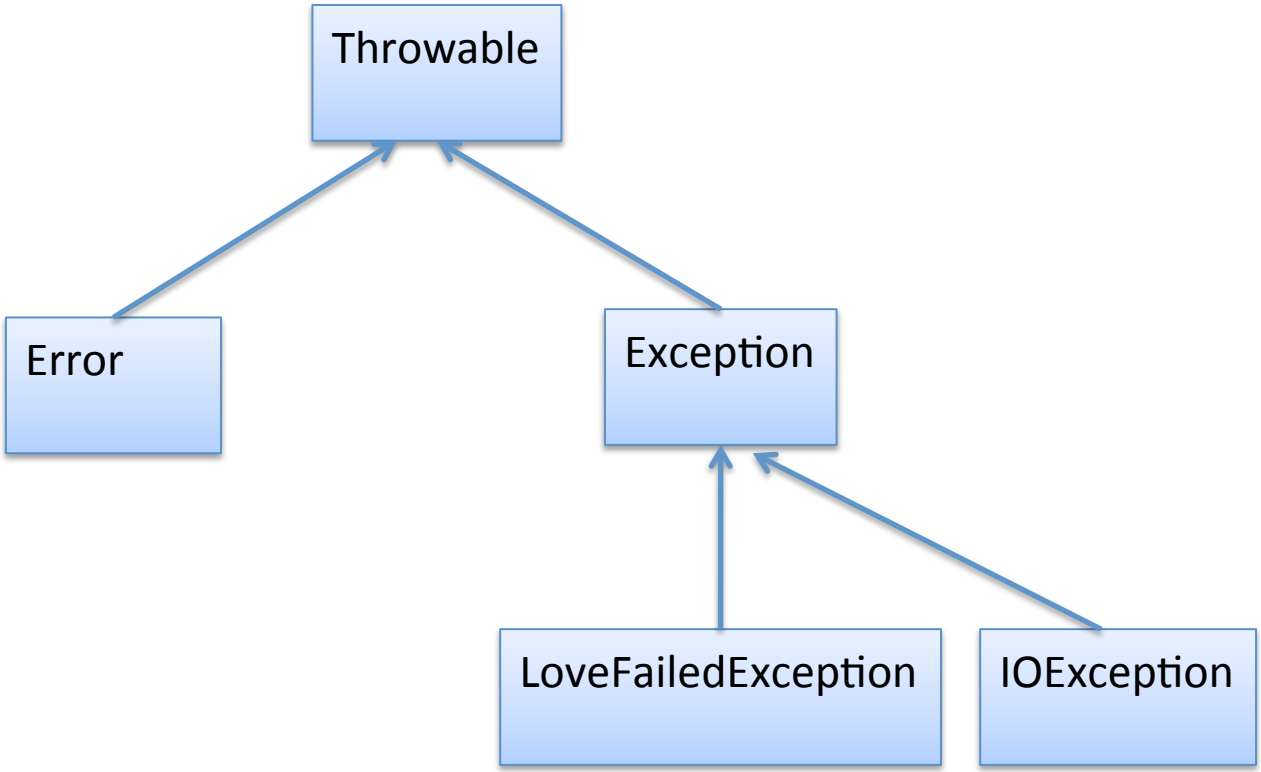
```
public class TestPayIncreasePoly {

  public static void main(String[] args) {

      Payable workers[] = new Payable[3];

            workers[0] = new Employee("John");
            workers[1] = new Contractor("Mary");
            workers[2] = new Employee("Steve");

            for (Payable p: workers){
              p.increasePay(30);
            }
  }
}
```

Assumption: both Employee and Contractor implement Payable that declares a method increasePay().

UI in Java can be programmed either using Swing library or in its modern wrapper JavaFX.

JavaFX want to compete with Flex and AIR.

Error Handling is Enforced in Java



Subclasses of Exception are called *checked* exceptions and must be handled in your code.

Subclasses of the class Error are fatal errors. They are called *unchecked exceptions*.

Java Collection Framework

- Classes located in the packages **java.util** and **java.util.concurrent** are often called Java collections.

- ArrayList, HashMap, Hashtable, Iterator, Properties, Collections ….

- Collections store Java objects – no primitives allowed.

Populating an ArrayList

```
ArrayList customers = new ArrayList();

Customer cust1 = new Customer("David","Lee");
customers.add(cust1);

Customer cust2 = new Customer("Ringo","Starr");
customers.add(cust2);
```

add() doesn't copy instance of the Customer into the customers collection, it just adds the memory address of the Customer being added.

You can specify initial size of ArrayList by using constructor with the argument:

```
ArrayList customers = new ArrayList(10);
```

Hashtable and Hashmap store key-value pairs

```
Customer cust = new Customer("David", "Lee");
Order ord = new Order(123, 500, "IBM");
Portfolio port = new Portfolio(123);

Hashtable data = new Hashtable();

data.put("Customer", cust);
data.put("Order", ord);
data.put("Portfolio", port);
```

Getting the object by key:

```
Order myOrder = (Order) data.get("Order");
```

Hashtable is synchronized, but Hashmap is not. You'll understand the difference after learning about threads and concurrent access.

Hashtable is not used very often. It has better replacements in the java.concurrent package.

Hashtable and Hashmap are for key-value pairs

```
Customer cust = new Customer("David", "Lee");
Order ord = new Order(123, 500, "IBM");
Portfolio port = new Portfolio(123);

Hashtable data = new Hashtable();

data.put("Customer", cust);
data.put("Order", ord);
data.put("Portfolio", port);
```

Retrieving an object by key:

```
Order myOrder = (Order) data.get("Order");
```

Hashtable is synchronized, but Hashmap is not. You'll understand the difference after learning about multi-threading and concurrent access.

Hashtable is not used very often. It has better replacements in the java.concurrent package.

Generics - Parameterized Data Types

Classes can have parameters – they are called *generics*.

ArrayList is a kitchen sink–like storage that can hold any object.
Getting an error during compilation is better than getting run-time cast exceptions.

```
ArrayList<Customer> customers = new ArrayList<>();

Customer cust1 = new Customer("David","Lee");
customers.add(cust1);

Customer cust2 = new Customer("Ringo","Starr");
customers.add(cust2);

Order ord1= new Order();
customers.add(ord1); // Compiler error because of <Customer>
```

Demo of Generics

Intro to Multi-Threading

- A program may need to execute some tasks **concurrently**, e.g. get market news and the user's portfolio data.
- Concurrent means parallel execution
- A Java program is a process.
- A *thread* is a light-weight process
- One Java program can start (*spawn*) multiple threads.

Intro to Multi-Threading (cont.)

- One server instance can process multiple clients' request by spawning multiple threads of execution (one per client).
- My MacBook Pro has 4 CPUs. Tasks can run in parallel.
- Even on a single-CPU machine you can benefit from the multi-threading – one thread needs CPU, the other waits for the user's input, the third one works with files.

The class Thread

```
public class MarketNews extends Thread {
 public MarketNews (String threadName) {
      super(threadName);  // name your thread
   }

  public void run() {
     System.out.println(
        "The stock market is improving!");
    }
}
```

```
public class Portfolio extends Thread {
   public Portfolio (String threadName) {
      super(threadName);
   }

   public void run() {
     System.out.println(
            "You have 500 shares of IBM ");
    }
}
```

```
public class TestThreads {
   public static void main(String args[]){
     MarketNews mn = new MarketNews("Market News");
     mn.start();

     Portfolio p = new Portfolio("Portfolio data");
     p.start();
     System.out.println( "TestThreads is finished");
   }
}
```

Interface Runnable

```
public class MarketNews2 implements Runnable {
    public void run() {
      System.out.println(
                  "The stock market is improving!");
    }
}
```

```
public class Portfolio2
                    implements Runnable {
   public void run() {
     System.out.println(
       "You have 500 shares of IBM ");
    }
}
```

```
public class TestThreads2 {
   public static void main(String args[]){

    MarketNews2 mn2 = new MarketNews2();
    Thread mn = new Thread(mn2,”Market News”);
    mn.start();

    Runnable port2 = new Portfolio2();
    Thread p = new Thread(port2, “Portfolio Data”);
    p.start();

    System.out.println( "TestThreads2 is finished”);
   }
}
```

Sleeping Threads

```
public class MarketNews3 extends Thread {
  public MarketNews3 (String str) {
   super(str);
  }

 public void run() {
  try{
    for (int i=0; i<10;i++){
     sleep (1000);  // sleep for 1 second
     System.out.println( "The market is improving " + i);
    }
  }catch(InterruptedException e ){
     System.out.println(Thread.currentThread().getName()
                                + e.toString());
  }
 }
}
```

```
public class Portfolio3 extends Thread {
  public Portfolio3 (String str) {
      super(str);
  }

  public void run() {
   try{
    for (int i=0; i<10;i++){
     sleep (700);    // Sleep for 700 milliseconds
     System.out.println( "You have " +  (500 + i) +
                                 " shares of IBM");
    }
  }catch(InterruptedException e ){
    System.out.println(Thread.currentThread().getName()
                                 + e.toString());
  }
 }
```

```
public class TestThreads3 {

   public static void main(String args[]){

       MarketNews3 mn = new MarketNews3("Market News");
       mn.start();

       Portfolio3 p = new Portfolio3("Portfolio data");
       p.start();

       System.out.println( "The main method of TestThreads3 is finished");
   }
}
```

Thread Synchronization and Race Conditions

- A *race condition* may happen when multiple threads need to modify the same program resource at the same time (concurrently).

- A classic example: a husband and wife are trying to withdraw cash from different ATMs at the same time.

- To prevent race conditions Java always offered the keyword synchronized. The preferred way though is the class java.util.concurrent.locks.ReentrantLock.

- The synchronized places a lock (a monitor) on an important object or piece of code to make sure that only one thread at a time will have access to it.

Minimize the locking periods

```
class ATMProcessor extends Thread{
  …
 synchronized withdrawCash(int accountID, int amount){
   // Some thread-safe code goes here, i.e. reading from
   // a file or a database
     …
  boolean allowTransaction = validateWithdrawal(accountID,
                                        amount);
  if (allowTransaction){
     updateBalance(accountID, amount, "Withraw");
  }
  else {
     System.out.println("Not enough money on the account");
  }
 }
}
```

Synchronizing the entire method

Synchronizing the code block

```
class ATMProcessor extends Thread{
  …
 withdrawCash(int accountID, int amount){
   // Some thread-safe code goes here, i.e. reading from
   // a file or a database
   …
  synchronized(this) {
   if (allowTransaction){
    updateBalance(accountID, amount, "Withdraw");
   }
   else {
    System.out.println(
                        "Not  enough money on the account");
   }
  }
 }
}
```

Executor Framework

Creating threads by subclassing Thread or implementing Runnable has shortcomings:

1. The method run() cannot return a value.

2. An application may spawn so many threads that it can take up all the system resources.

You can overcome the first shortcoming by using the Callable interface, and the second one by using classes from the *Executor framework*.

The Executors class spawns the threads from Runnable objects.

ExecutorService knows how to create Callable threads.

ScheduledExecutorService allows you to schedule threads for future execution.

Demo of Threads

Java Annotations

- *Metadata is the data about your data, a document, or any other artifact.*

- Program's metadata is the data about your code. Any Java class has its metadata embedded, and you can write a program that "asks" another class, "What methods do you have?"

- Java allows you to declare your own custom *annotations* and define your own processing rules that will route the execution of your program and produce configuration files, additional code, deployment descriptors, and more.

Predefined Java Annotations

- There are about a dozen of predefined annotations in Java SE, the packages java.lang, java.lang.annotation, and javax.annotation.

- Some of these annotations are used by the compiler (@Override, @SuppressWarning, @Deprecated, @Target, @Retention, @Documented, and @Inherited); some are used by the Java SE run-time or third-party run times and indicate methods that have to be invoked in a certain order (@PostConstruct, @PreDestroy), or mark code that was generated by third-party tools (@Generated).

- In Java EE annotations are being used everywhere

@Override

```
public class NJTax extends Tax {

      @override
       public double calcTax() {
            double  stateTax=0;
            if (grossIncome < 30000) {
              stateTax=grossIncome*0.05;
            }
            else{
              stateTax= grossIncome*0.06;
            }

            return stateTax - 500;
       }
}
```

```
class Tax{
   double grossIncome;
   String state;
   int dependents;

   public double calcTax() {
      double  stateTax=0;
      if (grossIncome < 30000) {
        stateTax=grossIncome*0.05;
      }
      else{
        stateTax= grossIncome*0.06;
      }
        return stateTax;
  }
}
```

Try

@Override public double calcTax(String something)

Compiler gives an error:

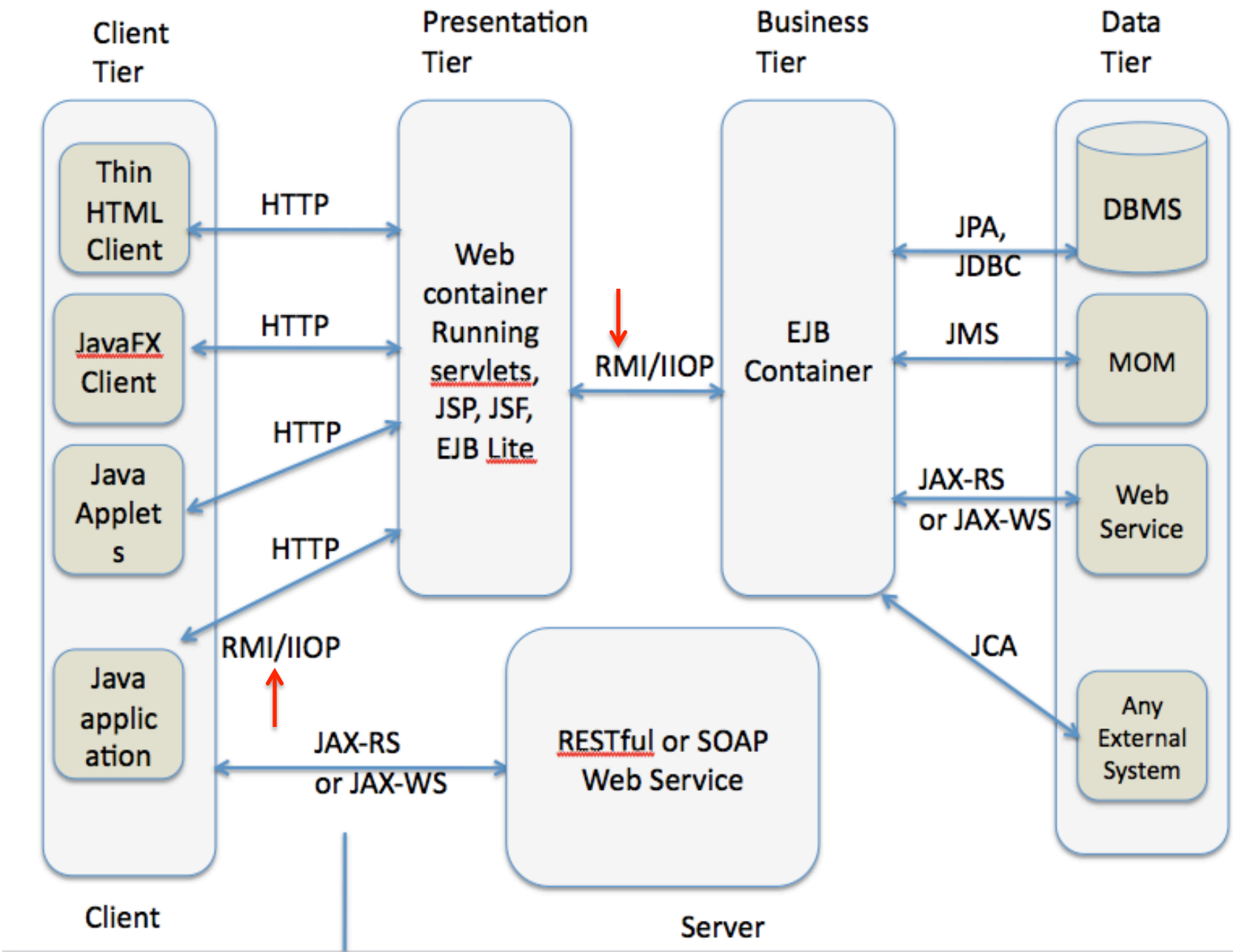
The method calcTax(String) of type NJTax must override or implement a supertype method

Custom Annotations

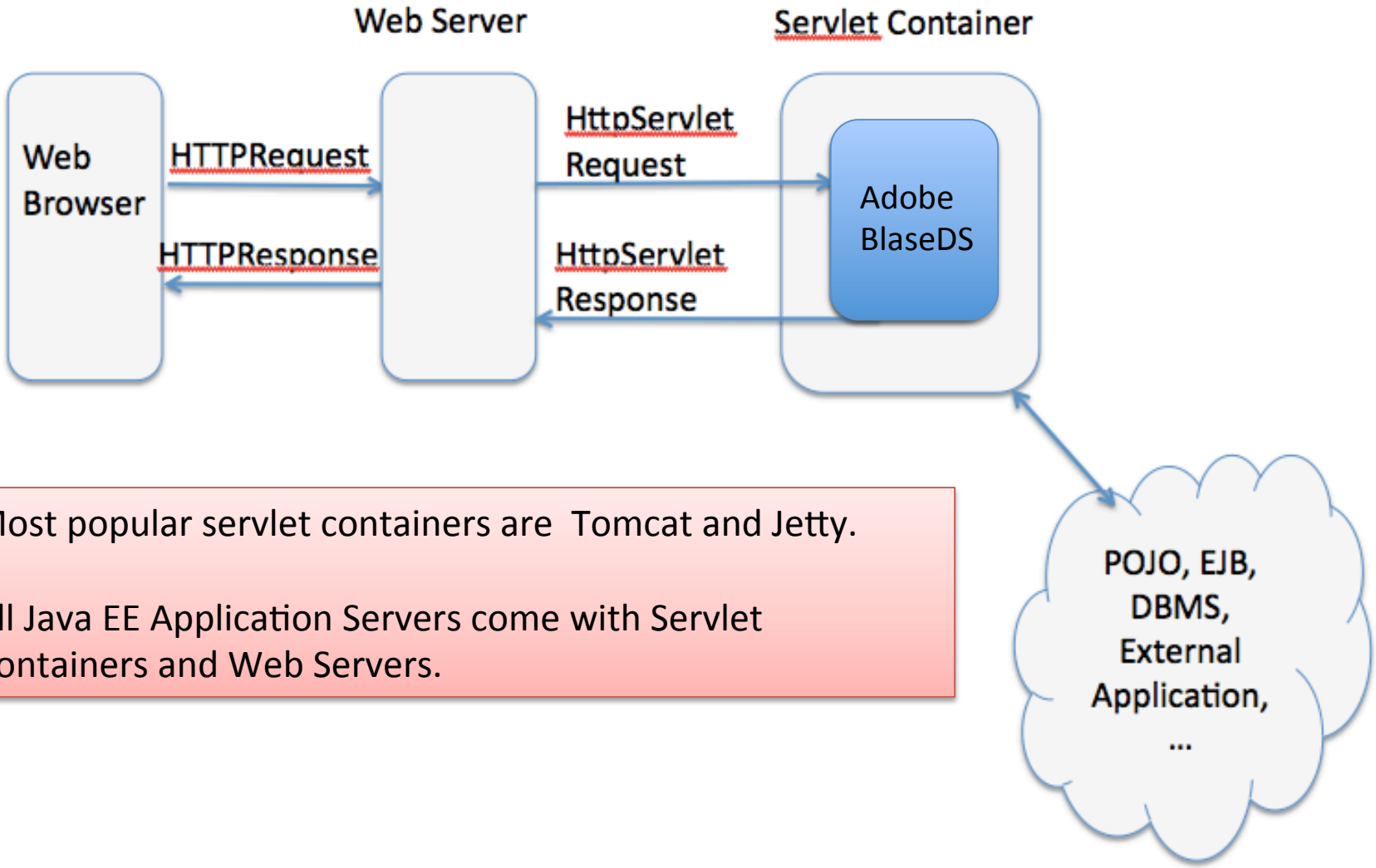
Java has a mechanism for creation of your own annotations and annotation processors.

For example, you may create an annotation that will allow other programmers to mark class methods with an SQL statement to be executed during the run time.

Java EE 6 Overview



Web applications with Servlets



Most popular servlet containers are Tomcat and Jetty.

All Java EE Application Servers come with Servlet Containers and Web Servers.

How to write a servlet

- To create a servlet, write a class that extends from HTTPServlet and annotate it with @WebServlet annotation.

- The servlet receives client's request and directs it to one of the methods of your servlet that you have to override, e.g. doGet(), doPost() et al.

Your First Servlet

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;

@WebServlet(urlPatterns="/books", name="FindBooks" )
public class FindBooks extends HttpServlet {

  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws ServletException {

      // The code processing request goes here
      // The resulting Web page will be sent back via the
      // I/O stream that response variable contains

      PrintWriter out = response.getWriter();
      out.println("Hello from FindBooks");
  }
}
```

Deploying a servlet

The annotation @WebServlet is a place where you specify servlet deployment parameters.

```
@WebServlet(urlPatterns="/books", name="FindBooks"
```

Every application server or servlet container has a directory known as document root. It is used not only for servlet-based Web sites, but also for deploying static HTML files.

For example, if you put the HTML file TermAndConditions.html in a subfolder legal of document root in the server MyBooks.com, the users would need to direct their Web browser to http://www.mybooks.com/legal/TermAndConditions.html.

In GlassFish application server, the default document root is directory /glassfish/domains/domain1/docroot.

In Apache Tomcat it's the directory webapps.
If you are planning to create a servlet, its deployment directory will also be located in document root, but it will contain the subdirectories WEB-INF and META-INF.

Sample Directory Structure of a Deployed Servlet

```
document root dir
    WEB-INF
        classes
            com
                practicaljava
                lesson27
                    FindBooks.class
        lib
    META-INF
        manifest.mf
```

Demo of a Dynamic Web project in Eclipse with a servlet

1. Create a dynamic Web project lesson27 by selecting Eclipse menu File | New | Other | Web | Dynamic Web Project. Make sure that the target runtime is Tomcat.

2. Observe the folder WebContent in your project. This is your server-side deployment part.

3. Generate new Servlet class: right-click on the project name and select New | Servlet. Specify com.practicaljava.lesson27 as the name of the package and the FindBooks as the class name. Press Next and enter /book by editing the URL mapping field.

4. In the next window keep the defaults methods doGet() and doPost() and press Finish.

5. The source code of the FindBooks servlet will be generated.

See the next slide

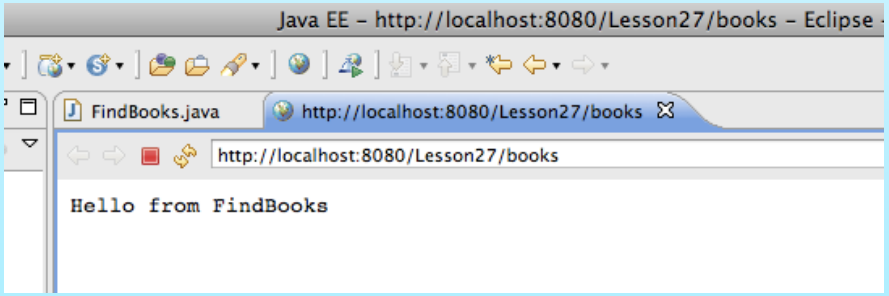
6. Add the following two lines in the method doGet():

```
PrintWriter out = response.getWriter();
out.println("Hello from FindBooks");
```

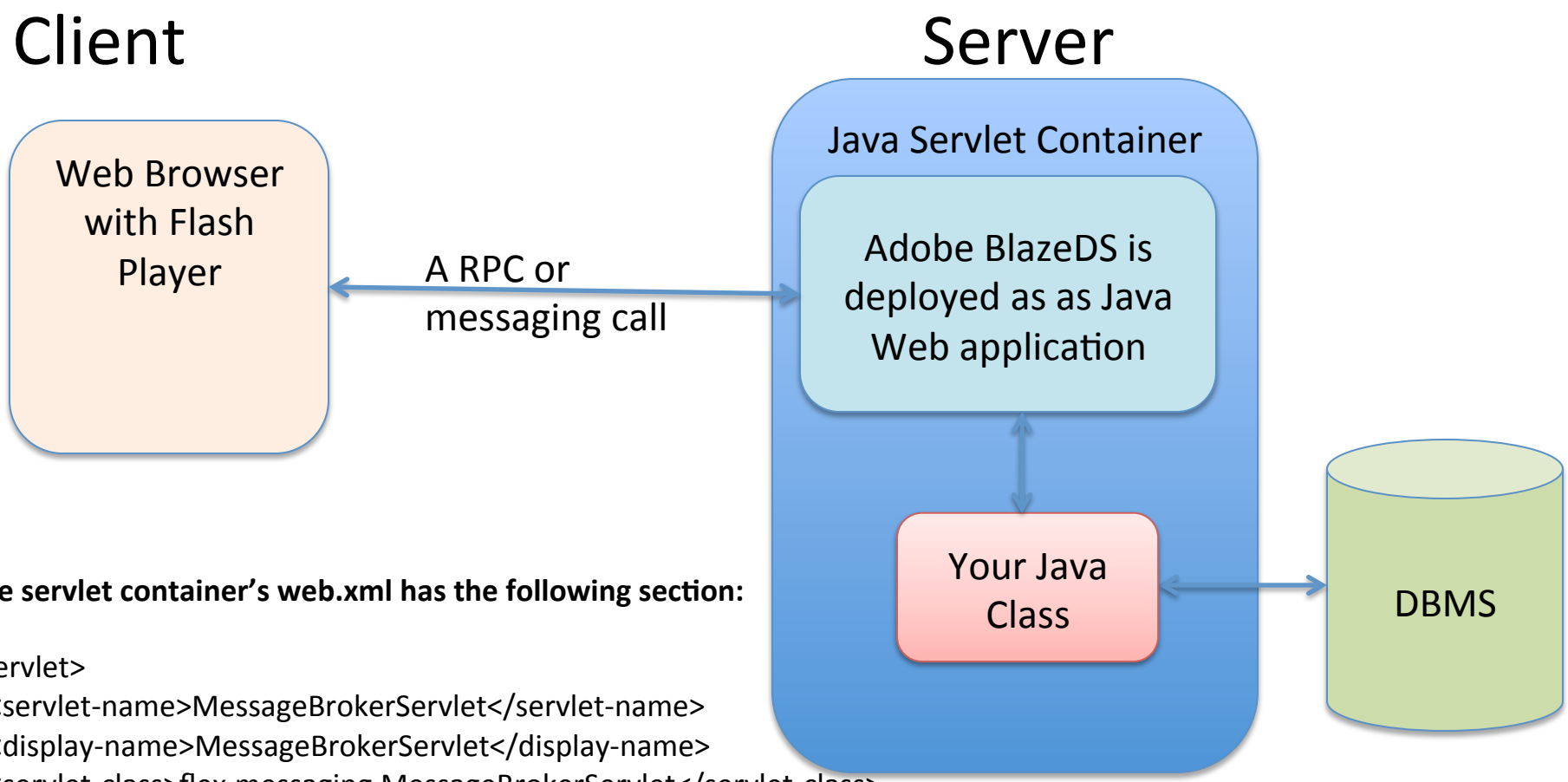
7. Import PrintWriter class

8. Deploy the servlet in Tomcat: open the Servers view, right-click on the server and select Add and Remove from the menu. Select lesson27 in the left panel and add it to the right one.

9. Run the servlet: right-click on FindBooks and select Run on Server. Eclipse will start its internal browser and display the following:



Flex-BlazeDS-Java-DBMS communications



The servlet container's web.xml has the following section:

```
<servlet>
  <servlet-name>MessageBrokerServlet</servlet-name>
  <display-name>MessageBrokerServlet</display-name>
  <servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>

  <init-param> <param-name>services.configuration.file</param-name>
   <param-value>/WEB-INF/flex/services-config.xml</param-value>
  </init-param>
 </servlet>
```

Configure *destination* in BlazeDS remoting-config.xml mapped to your java class.

Creating Flex/BlazeDS Project

Instructions below work in Eclipse IDE for JavaEE developers with installed Flash Builder plugin.

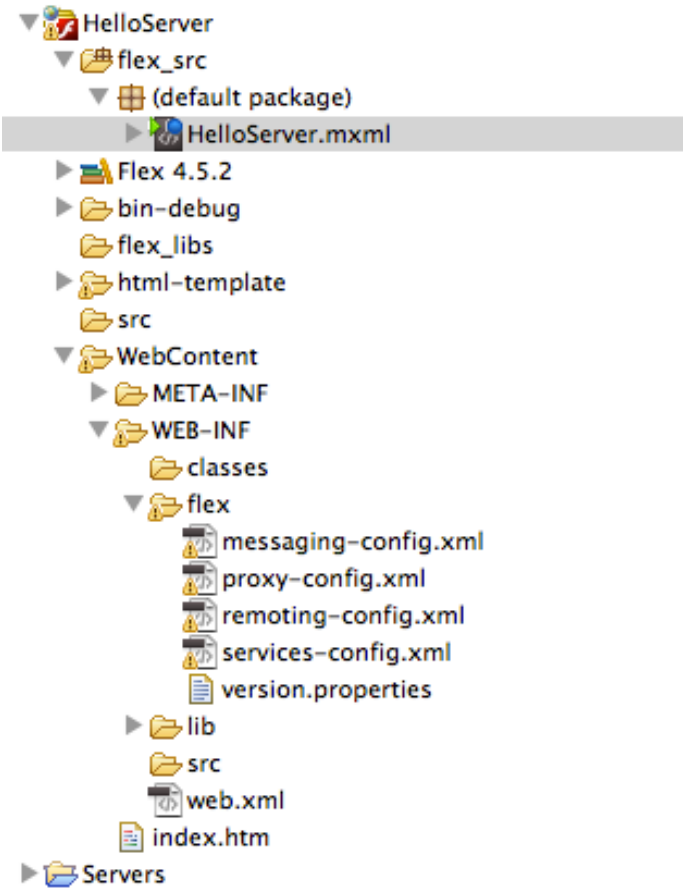
In Flash perspective select menu File | New | Flex Project and name it HelloServer. Press Next.

On the next popup window select Java as your application server type. Select BlazeDS radio button.

Note the checked box “Select combined Java/Flex project using WTP”

Select Apache Tomcat as your target runtime.

Click on the button Browse and select your downloaded *blazeds.war* file. Press Finish.



THANK YOU

Email: yfain@faratasystems.com
Twitter: @yfain