**STRING**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| contains | | Returns true if one string contains another. | | | | |
| endsWith | | Returns true if a string ends with a specified string. | | | | |
| indexOf | | Returns the index within a string at which the first occurrence of another string is found. Returns –1 if the string is not found. | | | | |
| lastIndexOf | | Returns the index within the invoking string at which the last occurrence of the specified string is found. Returns –1 if the string is not found. | | | | |
| startsWith | | Returns true if a string starts with a specified string. | | | | |
| compareTo | | | Compares one string to another. | | | |
| compareToIgnoreCase | | | Compares one string to another. Case differences are ignored. | | | |
| contentEquals | | | Compares a string to a specified character sequence. | | | |
| equals | | | Returns true if two strings contain the same character sequence. | | | |
| equalsIgnoreCase | | | Returns true if two strings contain the same character sequence. Case differences are ignored. | | | |
| matches | | | Returns true if a string matches a specified regular expression. | | | |
| regionMatches | | | Returns true if the specified region of one string matches the specified region of another. | | | |
| replace | | | | Replaces all occurrences of one character or substring with another. | | |
| replaceFirst | | | | Replaces the first character sequence that matches a specified regular expression. | | |
| replaceAll | | | | Replaces all character sequences that match a specified regular expression. | | |
| split | | | | Splits a string into substrings based on a sequence of delimiters specified by a regular expression. | | |
| substring | | | | Returns the specified portion of a string. | | |
| trim | | | | Returns a string in which the leading and trailing sp | | |
| toLowerCase | | | | | Converts a string to lowercase. | | |
| toUpperCase | | | | | Converts a string to uppercase. | | |

[^wxyz] matches any character *except* w, x, y, or z

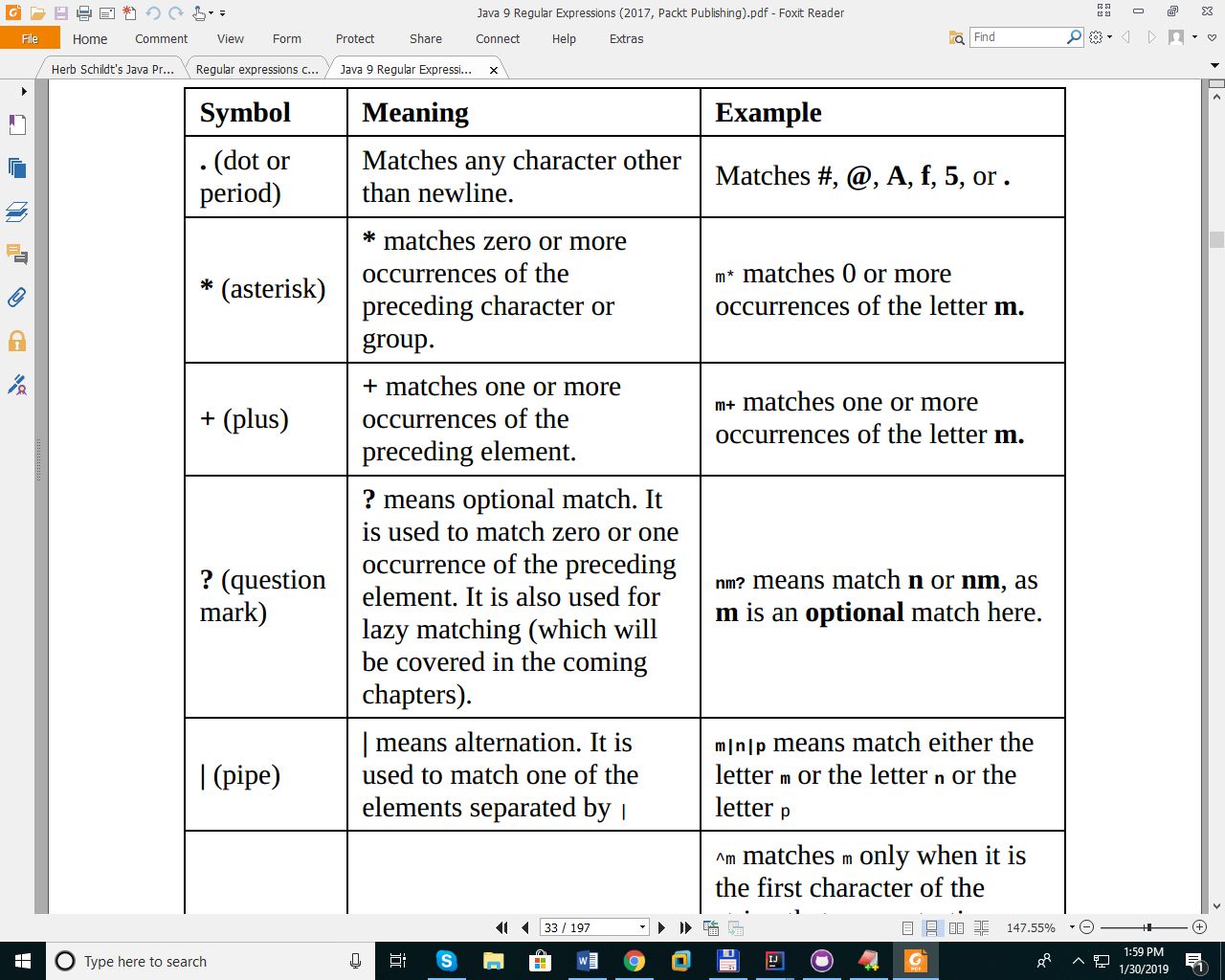
[1-9] match the digits 1 through 9

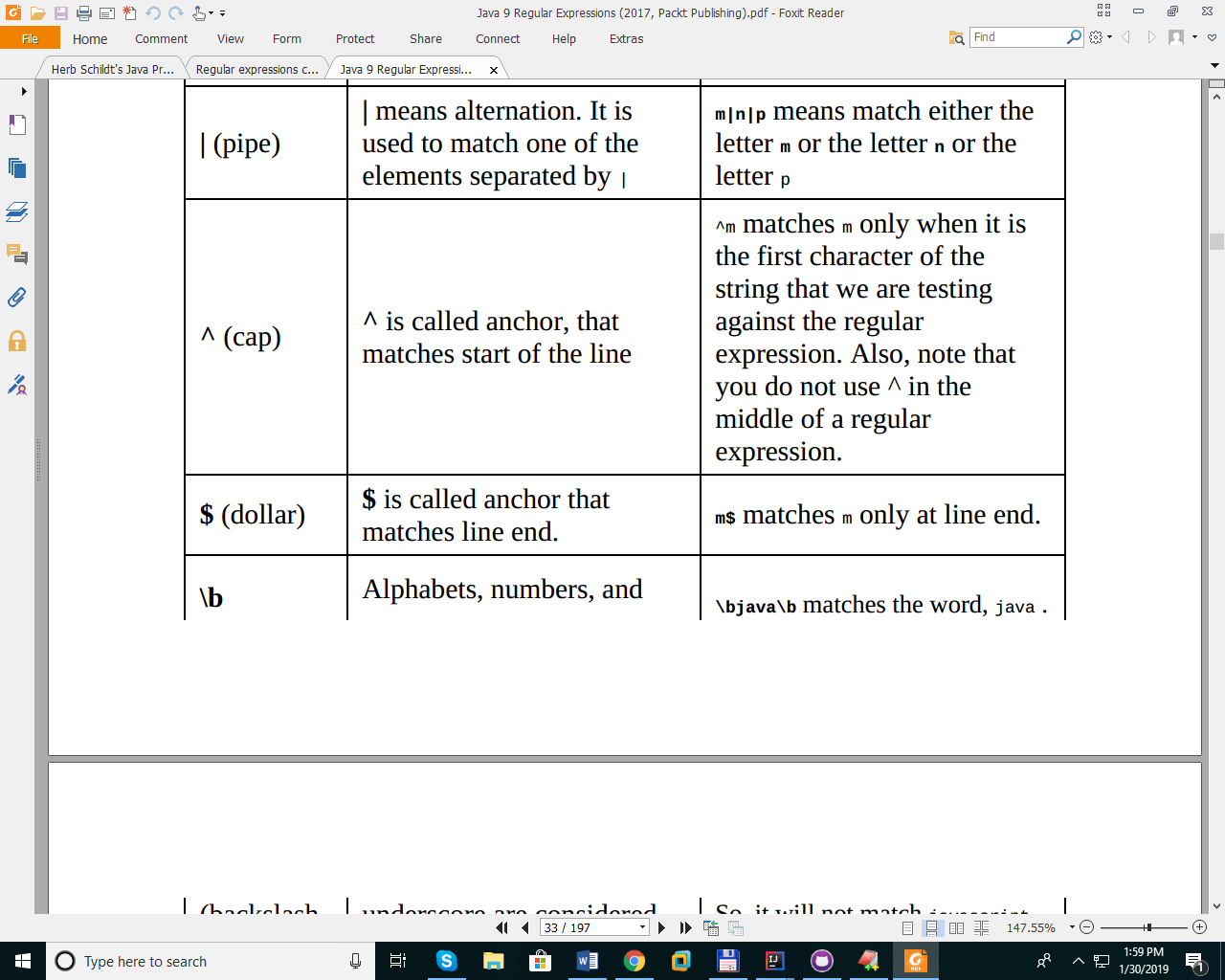
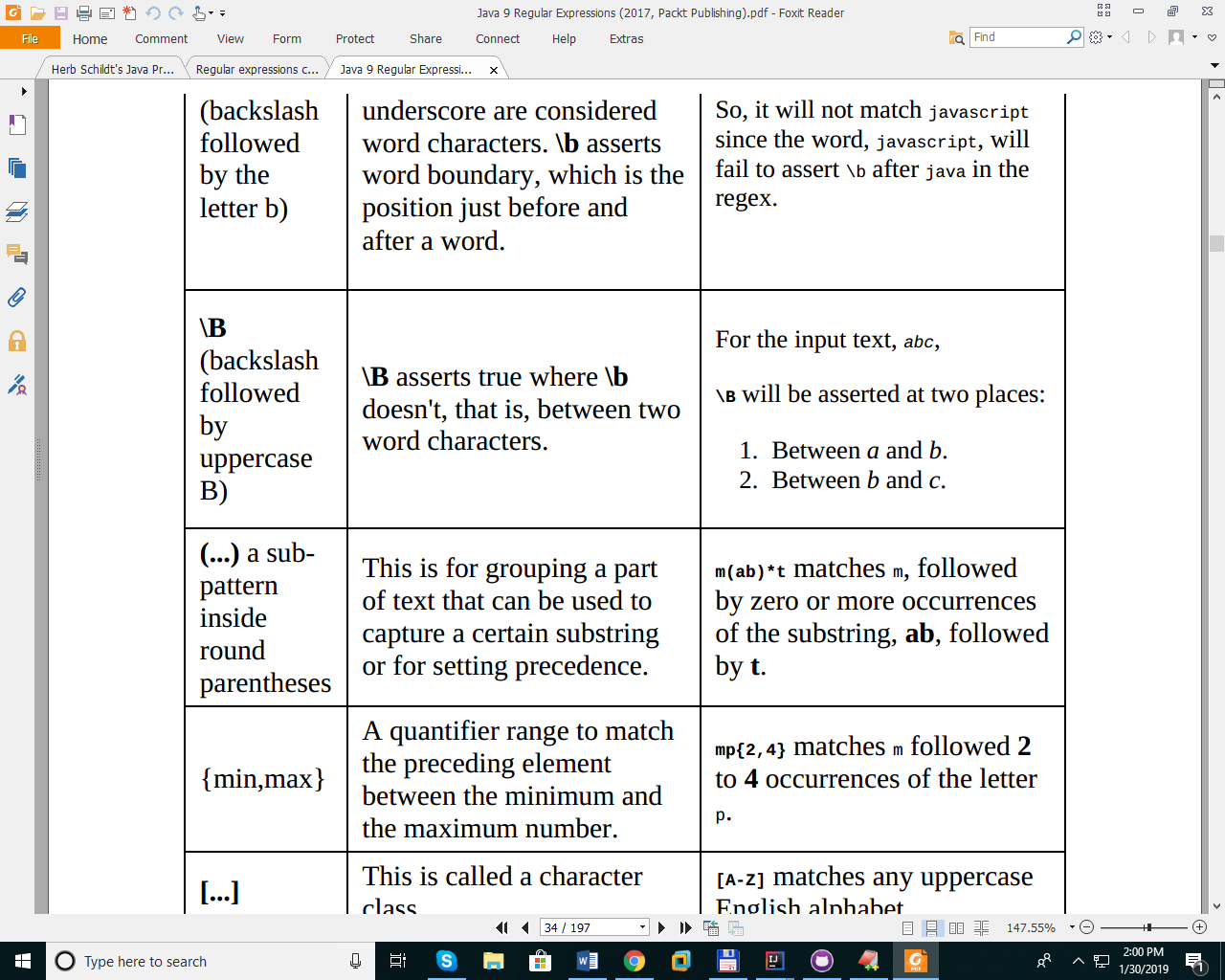
\w && [^A-Z]] this creates a set that matches all word characters except for uppercase letters.

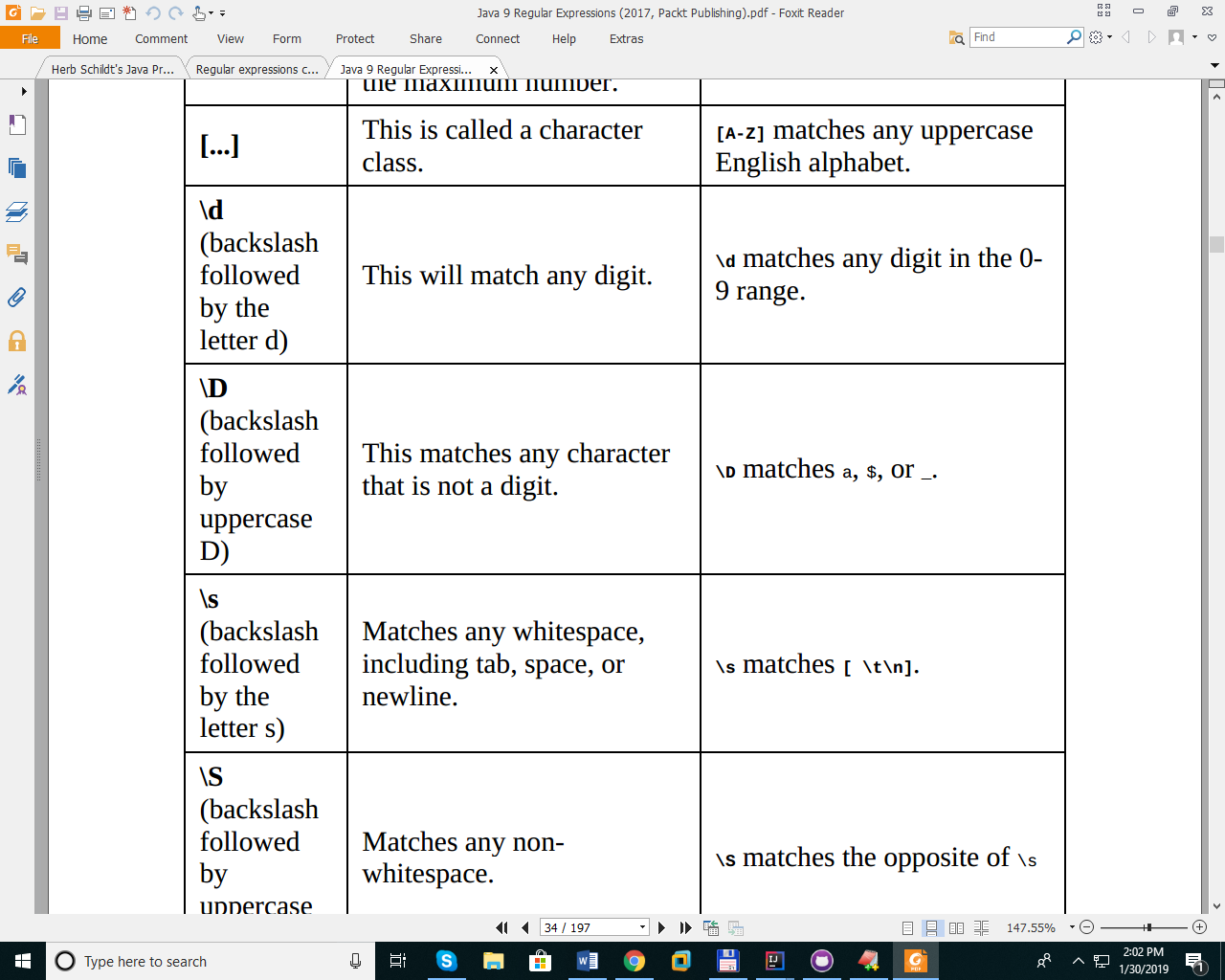
+ Match one or more.  
\* Match zero or more.  
? Match zero or one.

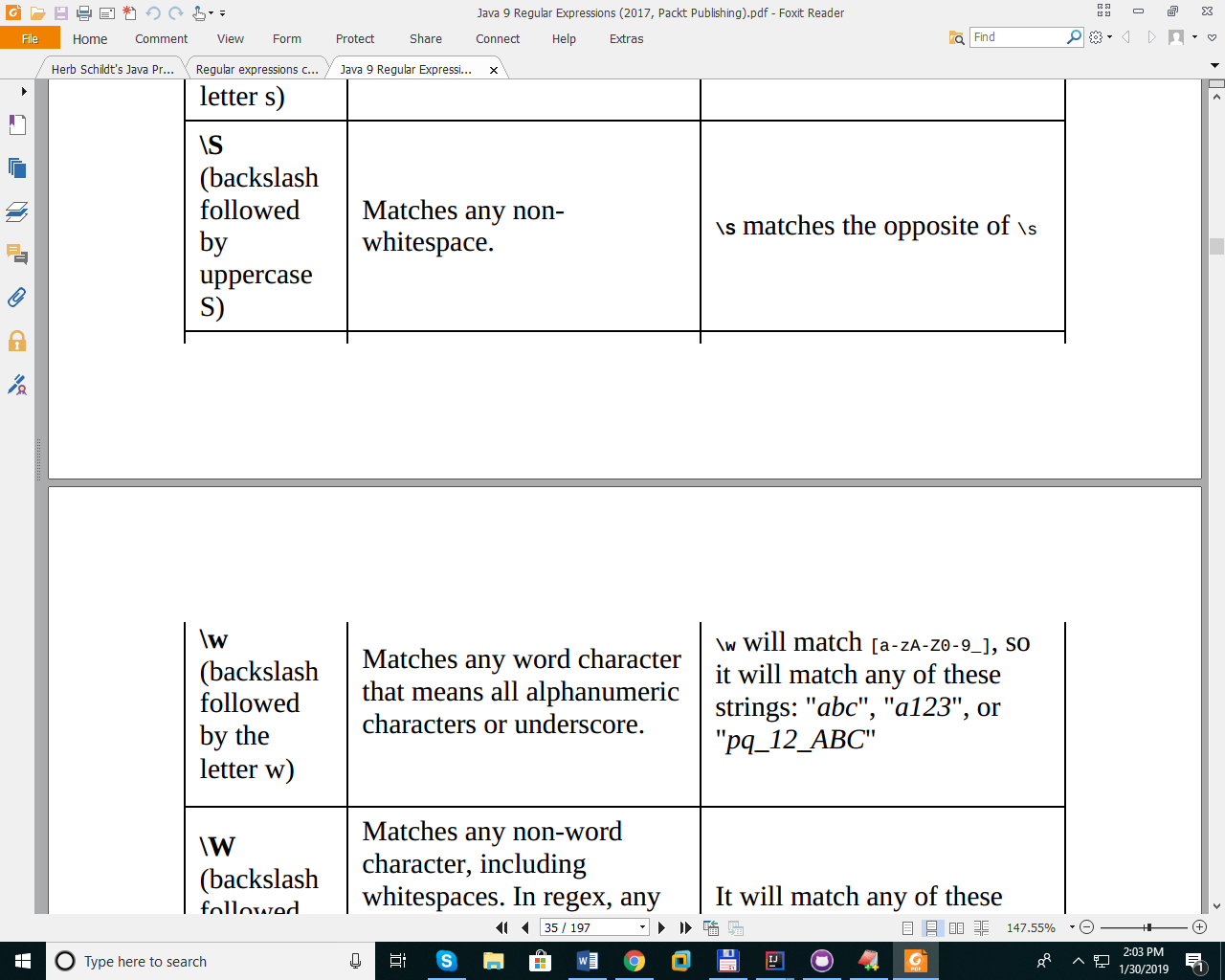
For example, x+ will match one or more x’s, such as "x", "xx", "xxx", and so on. The  
pattern .\* will match any character zero or more times. The pattern ,? will match zero or  
one comma.

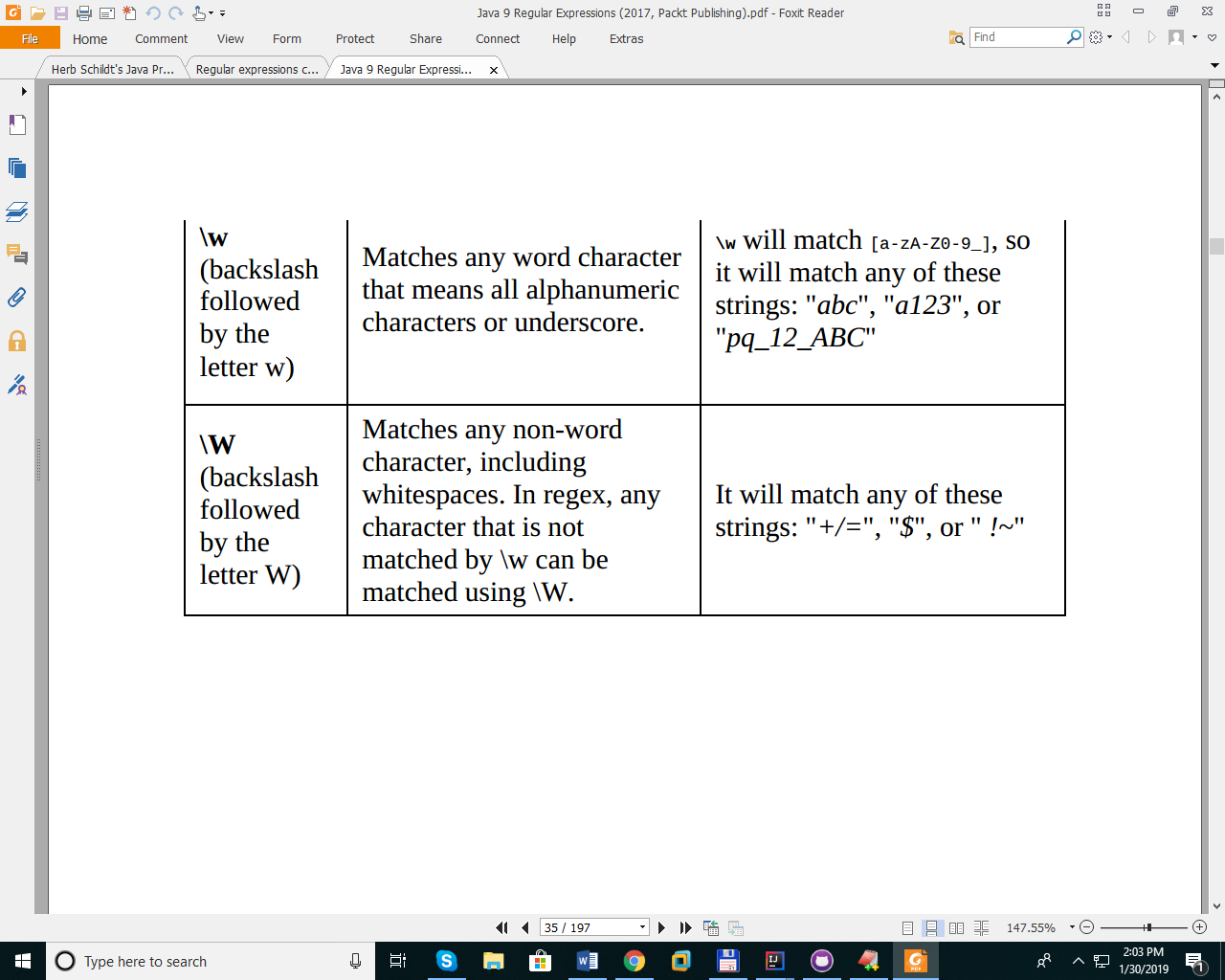
|  |  |
| --- | --- |
| Predefined Class | Matches |
| \d | The digits 0 through 9 |
| \D | All non-digits |
| \s | Whitespace |
| \S | All non-whitespace |
| \w | Characters that can be part of a word. In Java, these are the upper- and lowercase letters, the digits 0 through 9, and the underscore. These are commonly referred to as *word characters.* |
| \W | All non-word characters |









ab\*c This will match a, followed by zero or more b, followed by c.  
ab+c This will match a followed by one or more b, followed by c

ab?c This will match a followed by zero or one b, followed by c. Thus, it will match both abc or ac

^abc$ This will match abc in a line, and the line must not have anything other than the string abc due to the use of the start and end anchors on either side of the regex

a(bc)\*z This will match a, followed by zero or more occurrences of the string bc, followed by z. Thus, it will match the following strings: az, abcz, abcbcz, abcbcbcz, and so on

ab{1,3}c This will match a, followed by one to three occurrences of b, followed by c. Thus, it will match following strings: abc, abbc, and abbbc.

red|blue This will match either the string red or the string blue.

\b(cat|dog)\b This will match either the string cat or the string dog, ensuring both cat and dog must be complete words; thus, it will **fail** the match if the input is cats or dogs

[0-9] This is a character class with a character range. The preceding example will match a digit between 0 and 9.

[a-zA-Z0-9] This is a character class with a character range. The preceding example will match any alpha-numeric character.

^\d+$ This regex will match an input containing only one or more digits

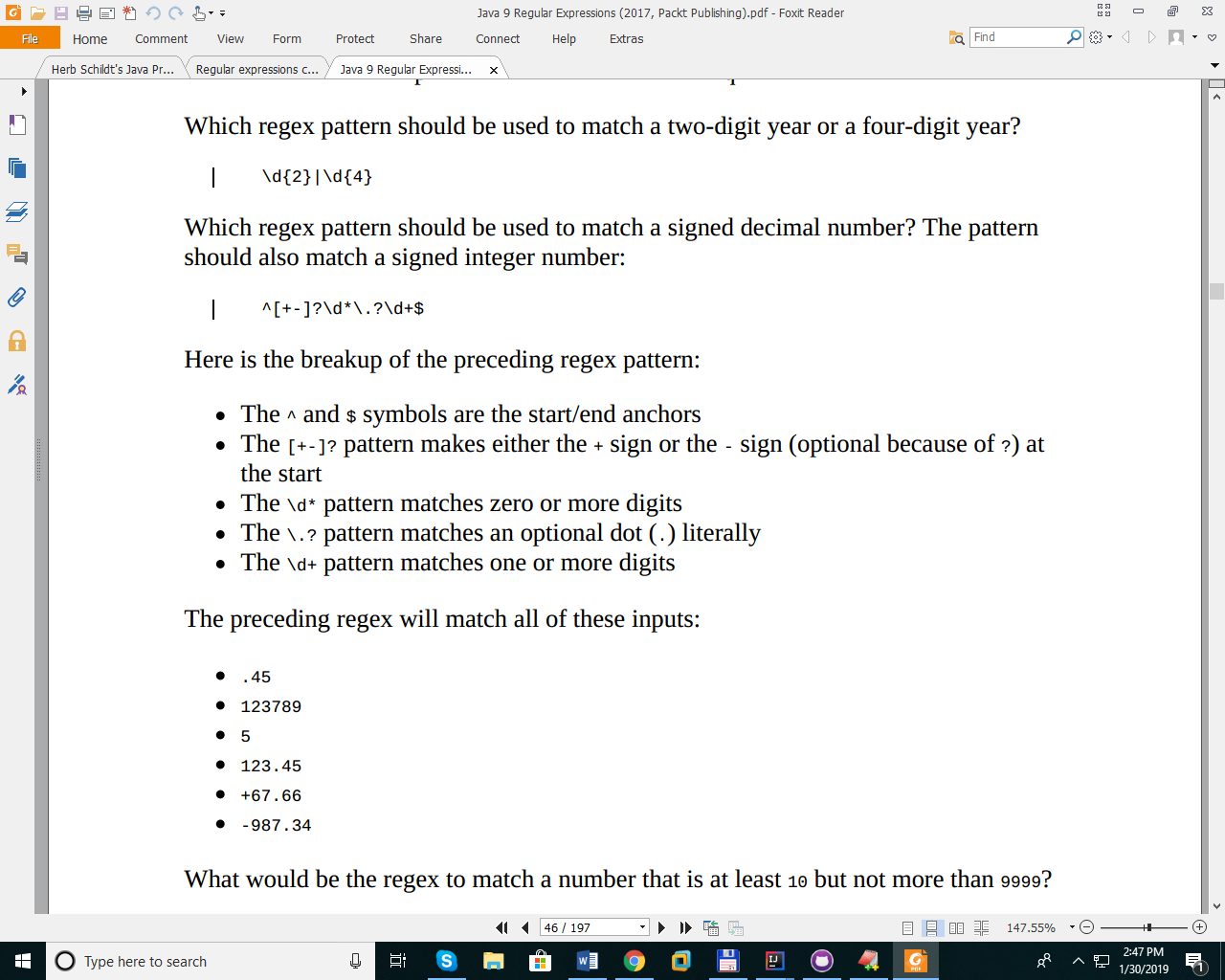
^\d{4,8}$ This regex will allow an input containing four to eight digits only. For example, 1234, 12345, 123456, and 12345678 are valid inputs

^\d\D\d$ This regex not only allows only one digit at the start and end but also enforces that between these two digits there must be one non-digit character. For example, 1-5, 3:8, 8X2, and so on are valid inputs

^\d+\.\d+$ This regex matches a floating point number. For example, 1.23, 1548.567, and 7876554.344 are valid inputs

.+ This matches any character one or more times. For example, qwqewe, 12233, or f5^h\_=!bg are all valid inputs

^\w+\s+\w+$ This matches a word, followed by one or more whitespaces, followed by another word in an input. For example, hello word, John Smith, and United Kingdom will be matched using this regex



**STREAM**

**https://www.geeksforgeeks.org/stream-in-java/**

**Intermediate Operations:**

1. **map:**The map method is used to map the items in the collection to other objects according to the Predicate passed as argument.  
   List number = Arrays.asList(2,3,4,5);  
   List square = number.stream().map(x->x\*x).collect(Collectors.toList());
2. **filter:** The filter method is used to select elements as per the Predicate passed as argument.  
   List names = Arrays.asList("Reflection","Collection","Stream");  
   List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
3. **sorted:** The sorted method is used to sort the stream.  
   List names = Arrays.asList("Reflection","Collection","Stream");  
   List result = names.stream().sorted().collect(Collectors.toList());

**Terminal Operations:**

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.  
   List number = Arrays.asList(2,3,4,5,3);  
   Set square = number.stream().map(x->x\*x).collect(Collectors.toSet());
2. **forEach:** The forEach method is used to iterate through every element of the stream.  
   List number = Arrays.asList(2,3,4,5);  
   number.stream().map(x->x\*x).forEach(y->System.out.println(y));
3. **reduce:** The reduce method is used to reduce the elements of a stream to a single value.  
   The reduce method takes a BinaryOperator as a parameter.

List number = Arrays.asList(2,3,4,5);  
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

|  |
| --- |
| //a simple program to demonstrate the use of stream in java  **import** java.util.\*;  **import** java.util.stream.\*;    **class** Demo  {  **public** **static** **void** main(String args[])    {        // create a list of integers      List<Integer> number = Arrays.asList(2,3,4,5);        // demonstration of map method      List<Integer> square = number.stream().map(x -> x\*x).                             collect(Collectors.toList());      System.out.println(square);        // create a list of String      List<String> names =                  Arrays.asList("Reflection","Collection","Stream");        // demonstration of filter method      List<String> result = names.stream().filter(s->s.startsWith("S")).                            collect(Collectors.toList());      System.out.println(result);        // demonstration of sorted method      List<String> show =              names.stream().sorted().collect(Collectors.toList());      System.out.println(show);        // create a list of integers      List<Integer> numbers = Arrays.asList(2,3,4,5,2);        // collect method returns a set      Set<Integer> squareSet =           numbers.stream().map(x->x\*x).collect(Collectors.toSet());      System.out.println(squareSet);        // demonstration of forEach method      number.stream().map(x->x\*x).forEach(y->System.out.println(y));        // demonstration of reduce method  **int** even =         number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);        System.out.println(even);    }  } |

//remove item from a list  
  
List<String> fruits = new ArrayList<>();  
fruits.add("Apple");  
fruits.add("Banana");  
fruits.add("Stawberry");  
  
Iterator<String> fruitIterator = fruits.iterator();  
  
while(fruitIterator.hasNext()) {  
 if(fruitIterator.next().contains("Apple")) {  
 fruitIterator.remove();  
 }  
}

System.*out*.println (fruits);

// defference between List - Set  
  
List<String> data = Arrays.*asList*("ab","fc","cd","ab","bc","cd");  
  
List<String> list = new ArrayList<>(data);// will add data as is  
Set<String> set1 = new HashSet<>(data); // will add data keeping only unique values  
Set<String> set2 = new TreeSet<>(data); // will add data keeping unique values and sorting  
Set<String> set3 = new LinkedHashSet<>(data); // will add data keepig only unique values and preserving the original order  
  
System.*out*.println(list);  
System.*out*.println(set1);  
System.*out*.println(set2);  
System.*out*.println(set3);

**UnModiFiableList -->ReadOnly**

public class UnModifiablList {  
  
 private List<Integer> intList = new ArrayList<>();  
  
 public void addValue(Integer value) {  
 intList.add(value);  
 }  
  
 public List<Integer> getIntList() {  
 return Collections.*unmodifiableList*(intList);

**Check Funtional Interface**

package Main;  
  
@FunctionalInterface  
public interface bar {  
 void barInt();  
 default int batDouble() { return 0};  
}

**GENERIC CLASS**

Unlike other identifiers, type parameters have no naming constraints. However their names are commonly the first letter of their purpose in upper case. (This is true even throughout the official JavaDocs.)  
Examples include T for "type", E for "element" and K/V for "key"/"value"

public class TestGeneric<T> {  
  
 private T value;  
  
 public void setValue(T value) {  
 this.value=value;  
 }  
  
 public T getValue(){  
 return this.value;  
 }  
}

**DELETE FILE**

File file = new File("d:\\InputData\\JavaTakeNote.docx");  
if(file.exists()) {  
 boolean deleteStatus = file.delete();  
 if(!deleteStatus) {   
 throw new IOException("Unable to delete file");  
 }  
} else {  
 throw new IOException("File does not exist !" );  
}

**Iterating on each file within a folder**

File file = new File("d:\\InputData");  
for(File seletedFile : file.listFiles()) {  
 String property = seletedFile.isFile() ? "file" :"directory";  
 System.*out*.println (property + " " + seletedFile.getAbsolutePath() );  
  
}

**RECURSE FOLDER**

public static void recurseFolder (String path) {  
 File file = new File(path);  
 for(File seletedFile : file.listFiles()) {  
 String property = seletedFile.isFile() ? "file" :"directory";  
 if(property.equals("directory")){  
 *recurseFolder*(seletedFile.getAbsolutePath());  
 }  
  
 System.*out*.println (property + " " + seletedFile.getAbsolutePath() );

}  
}

**COPY FILE**

public static void copyFile(String sourceFile, String desFile) {  
 File source = new File (sourceFile);  
 File des = new File(desFile);  
 try {  
 FileChannel sourceChanel = new FileInputStream(source).getChannel();  
 FileChannel desChanel = new FileOutputStream(des).getChannel();  
 sourceChanel.transferTo(0,sourceChanel.size(),desChanel);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

*// Read from baseReader, one line at a time*BufferedReader reader = new BufferedReader( baseReader );  
String line;  
while((line = reader.readLine()) != null) {  
*// Remember: System.out is a stream, not a writer!*System.out.println(line);  
}

**READ FILE USE SCANNER**

public static void readFile(String sourceFile) {  
 try (Scanner sc = new Scanner(new File(sourceFile))) {  
  
 while (sc.hasNextLine()) {  
 System.*out*.println(sc.nextLine());  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
}

public static void readWoedByWord(String sourceFile) {  
 try (Scanner sc = new Scanner(new File(sourceFile))) {  
  
 while (sc.hasNext()) {  
 System.*out*.println(sc.next());  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
}

**CREATE A NEW FOLDER**

public static void makeFolder() {  
 String path ="d:\\InputData\\NewFolder";  
 File file = new File (path);  
 file.mkdir();  
  
}

**WRITE TEXT FILE**

public static void writeLineToFiile() {  
 String pathFile = "D:\\log.txt";  
 String line ="Level log : fine";  
 try {  
 BufferedWriter wr = new BufferedWriter(new FileWriter(pathFile));  
 wr.write(line);  
 wr.close();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

**READ TEXT FILE**

public static void readFile() {  
  
 String pathFile = "d:\\InputData\\strorage.xml";  
 try {  
 BufferedReader br = new BufferedReader(new FileReader(pathFile));  
 String s =null;  
 while ((s=br.readLine())!=null){  
 System.*out*.println(s);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**DESIGN PATTERN**

These patterns are divided into three categories: creational, structural, and behavioral.

**Creational patterns**—Patterns that control object creation, initialization, and class selection. Singleton (“Singleton Pattern”) and factory ( “Factory Pattern”) are examples from this group.

**Behavioral patterns**—Patterns that control communication, messaging, and interaction between objects. The observer ( “Observer Pattern”) is an example from this group.

**Structural patterns**—Patterns that organize relationships between classes and objects, providing guidelines for combining and using related objects together to achieve desired behaviors. The decorator pattern ( “Decorator Pattern”) is a good example of a pattern from this group

**SINGLETON PATTERN**

**Ensure that only one instance of a class is created** and **Provide a global access point to the object**

package Main;  
  
public enum SingleTon {  
  
 *INSTANCE*;  
  
 private SingleTon () {}  
  
 public void f() {  
 System.*out*.println("CODE HERE");  
  
 }  
}

**READ TEXT DATA FROM WEB**

public static void readWeb() {  
  
 String url = "https://vnexpress.net/";  
 try {  
 HttpsURLConnection httpCon = (HttpsURLConnection)new URL(url).openConnection();  
 int responsecode = httpCon.getResponseCode();  
 if(responsecode>=200&&responsecode<=299) {  
 InputStream inputstream = httpCon.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(inputstream));  
 String line;  
  
 while ((line =br.readLine())!=null) {  
 System.*out*.println(line);  
 }  
  
 br.close();  
  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

**DATE FORMAT**

public static void printDateFormat() {  
  
 String format ="yyyy/MM/dd hh:mm:ssXXX";  
  
 Date date = Calendar.*getInstance*().getTime();  
  
 SimpleDateFormat simpleDateFormat = new SimpleDateFormat(format);  
  
 String formatedDate = simpleDateFormat.format(date);  
  
 System.*out*.println(date);  
 System.*out*.println(formatedDate);  
  
  
}

**REFLECTION**

public static void reflection() {  
  
 Class<String> clazz = String.class;  
 Constructor<?>[] constructor = clazz.getConstructors();  
 Method[]methods = clazz.getMethods();  
  
 for(Method method : methods) {  
 System.*out*.println(method);  
 }  
  
 for(Constructor c : constructor) {  
 System.*out*.println(c);  
 }  
  
}

**INVOKE FUNCTION USE REFLECTION**

public static void reflection() {  
  
 Class<String> clazz = String.class;  
 Constructor<?>[] constructor = clazz.getConstructors();  
 try {  
 Method method= clazz.getMethod("length");  
  
 System.*out*.println(method.invoke("Hello Reflection"));  
  
 } catch (NoSuchMethodException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (InvocationTargetException e) {  
 e.printStackTrace();  
 }

**CREATE RANDOM NUMBER**

public static void createRandomNumber() {  
  
 SecureRandom rng = new SecureRandom();  
 System.*out*.println(rng.nextInt());  
  
}

**PROCESS EXCUTE**

public static void process() {  
 try {  
 Process p = Runtime.*getRuntime*().exec("ipconfig");  
 InputStream inputstream = p.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(inputstream));  
 String line;  
 while ((line=br.readLine())!=null) {  
 System.*out*.println(line);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**PARSE XML**

public static Document parseXMLDocument(String fileName) {  
  
 File xmlFile = new File(fileName);  
 if(!xmlFile.exists()) {  
 System.*out*.println("File " + fileName + " does not exist");  
 }  
 try {  
 DocumentBuilder db = DocumentBuilderFactory.*newInstance*().newDocumentBuilder();  
 return db.parse(fileName);  
 } catch (ParserConfigurationException e) {  
 e.printStackTrace();  
 } catch (SAXException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
  
}

public void temp() {  
 String pathVC = System.*getProperty*("user.dir")+"\\datamodel\\vendorcerts\\test\\";  
  
 File files = new File(pathVC);  
 File []listVC = files.listFiles();  
 for(File vc : listVC) {  
 Document dom = XMLUtil.*parseXMLDocument*(vc.getAbsolutePath());  
 NodeList nListFacetType= dom.getElementsByTagName("FacetType");  
 Node nFacetType = nListFacetType.item(0);  
 Element eFacetType = (Element)nFacetType;  
 System.*out*.println(eFacetType.getAttribute("name")); }  
  
 System.*out*.println("Done");  
}

**GET ENVIRONMENT VARIABLEs**

public static void main(String []arg) throws IOException {  
  
 System.*out*.println(System.*getenv*());

System.*out*.println(System.*getenv*("PATH"));  
  
}

**GET PROPERTY INFO**

public static void main(String []arg) throws IOException {  
  
 System.*out*.println(System.*getProperties*());  
 System.*out*.println(System.*getProperty*("user.dir"));  
  
}

**READ INPUT STREAM OF A FILE**

public static void process() {  
  
 String input = "d:\\InputData\\AciStorage.json";  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream(input);  
 BufferedInputStream bis = new BufferedInputStream(fileInputStream);  
 byte[] buf = new byte[1024];  
 int temp = 0;  
 while((temp=bis.read(buf))!=-1) {  
 System.*out*.println(temp);  
 System.*out*.println(buf[2]);  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

**COVERT A FILE CONTENT TO BYTE**

public static void process() {  
  
 String input = "d:\\InputData\\AciStorage.json";  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream(input);  
 BufferedInputStream bis = new BufferedInputStream(fileInputStream);  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 byte[] buf = new byte[1024];  
 int len = 0;  
 while((len=bis.read(buf))!=-1) {  
 baos.write(buf, 0, len);  
 }  
 baos.flush();  
  
 byte [] result = baos.toByteArray();  
   
 System.*out*.println(result[2]);  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

public static void process() {  
  
 String input = "d:\\InputData\\temp.txt";  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream(input);  
 BufferedInputStream bis = new BufferedInputStream(fileInputStream);  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 byte[] buf = new byte[1024];  
 int len = 0;  
 while((len=bis.read(buf))!=-1) {  
 baos.write(buf, 0, len);  
 }  
 baos.flush();  
  
 byte [] result = baos.toByteArray();  
  
 String s = new String(result);  
  
 System.*out*.println(s.length());  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

**JMS (JAVA MESSAGE SERVICE)**

**Pom.XML**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>groupId</groupId>  
 <artifactId>JavaTest</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <dependencies>  
 <dependency>  
 <groupId>org.wso2.sp</groupId>  
 <artifactId>jms-producer</artifactId>  
 <version>4.2.0-M2</version>  
 <type>pom</type>  
 </dependency>  
 <dependency>  
 <groupId>javax.jms</groupId>  
 <artifactId>javax.jms-api</artifactId>  
 <version>2.0</version>  
 </dependency>  
 </dependencies>  
</project>

**CONNECT TO DATABASE**

Driver: Knows how to get a connection to the database  
Connection: Knows how to communicate with the database  
Statement: Knows how to run the SQL  
ResultSet: Knows what was returned by a SELECT query

1. **Config Setting.xml (download driver)**

**Add :**

**<repository>**

**<id>my-repo2</id>**

**<name>sql-lib</name>**

**<url>http://clojars.org/repo/</url>**

**</repository>**

1. **Code**

import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Connection conn = null;  
 Statement st = null;  
  
 try {  
 System.*out*.println("Begin");  
 Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
 conn = DriverManager.*getConnection*("jdbc:sqlserver://localhost:1433", "sa", "123456");  
 st = conn.createStatement();  
 String comand = "CREATE DATABASE VendorCertInfo";  
 st.executeUpdate(comand);  
 System.*out*.println("Done");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }

**GET DATA FROM DATABASE**

import java.sql.\*;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Connection conn = null;  
 Statement st = null;  
 ResultSet result =null;  
  
 try {  
 System.*out*.println("Begin");  
   
 Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
  
 String url= "jdbc:sqlserver://localhost:1433;databaseName=CertProject;user=sa;password=123456";  
  
 conn = DriverManager.*getConnection*(url);  
  
 st = conn.createStatement();  
  
 String comand = "SELECT \*FROM ColumExpression";  
  
 result = st.executeQuery(comand);  
  
 while (result.next()) {  
 String value = result.getString("Expression");  
 System.*out*.println(value);  
 }  
  
 System.*out*.println("Done");  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }

import java.sql.\*;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Connection conn = null;  
 PreparedStatement st = null;  
 ResultSet result =null;  
  
 try {  
 System.*out*.println("Begin");  
  
 Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
  
 String url= "jdbc:sqlserver://localhost:1433;databaseName=CertProject;user=sa;password=123456";  
  
 conn = DriverManager.*getConnection*(url);  
  
  
  
 String comand = "SELECT \*FROM ColumExpression";  
  
 st = conn.prepareStatement(comand);  
 result = st.executeQuery();  
  
 while (result.next()) {  
 String value = result.getString("Expression");  
 System.*out*.println(value);  
 }  
  
 System.*out*.println("Done");  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
}

**Lambda Expression (OCA\_ Oracle Certified Associate Java SE 8 Programmer I Study Guide\_ Exam 1Z0-808-Sybex (2014).pdf)**

public class Animal {  
  
 private String species;  
 private boolean canHop;  
 private boolean canSwim;  
 public Animal(String speciesName, boolean hopper, boolean swimmer) {  
 species = speciesName;  
 canHop = hopper;  
 canSwim = swimmer;  
 }  
 public boolean canHop() {  
 return canHop;  
 }  
  
 public boolean canSwim() {  
 return canSwim;  
 }  
  
 public String toString() {  
 return species;  
 }  
  
}

public interface CheckTrait {  
 boolean test(Animal a);  
}

import java.util.ArrayList;  
import java.util.List;  
  
public class Test {  
  
 public static void main(String []arg) {  
 List<Animal> animals = new ArrayList<Animal>(); // list of animals  
 animals.add(new Animal("fish", false, true));  
 animals.add(new Animal("kangaroo", true, false));  
 animals.add(new Animal("rabbit", true, false));  
 animals.add(new Animal("turtle", false, true));  
  
 *print*(animals,a->a.canSwim());  
  
 }  
  
 private static void print(List<Animal> animals, CheckTrait checker) {  
  
 for (Animal animal : animals) {  
 if (checker.test(animal))  
 System.*out*.print(animal + " ");  
 }  
 System.*out*.println();  
 }  
  
}

public class Animal {  
  
  
 public boolean canHop() {  
 System.*out*.println("This is can Hop");  
 return true;  
 }  
  
 public boolean canSwim() {  
 System.*out*.println("This is can Swim");  
 return false;  
 }  
  
  
  
}

public interface CheckTrait {  
 boolean test();  
}

import java.util.ArrayList;  
import java.util.List;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Animal a = new Animal();  
  
 *print*(()->a.canHop());  
  
 }  
  
 private static void print( CheckTrait checker) {  
  
 System.*out*.println (checker.test());  
  
 }  
  
}

**THREAD**

There are two ways to create threads: by extending the java.lang.Thread class and implementing the java.lang.Runnable interface.

1. Extending the java.lang.Thread class :

public class ThreadClass extends Thread {  
 public void run() {  
 for(int i =0; i< 100;++i) {  
 System.*out*.println("Thread Class : " + Thread.*currentThread*().getName() + " Value :" + i);  
 }  
 }  
}

ThreadClass t = new ThreadClass();  
t.start();

The run() method provides the entry point for the thread. It contains codes to be executed concurrently with other threads.

Note that the output may vary next time you run the program. This happens because the operating system may use a different interleaving pattern depending on its scheduling policy and other parameters.

2. Implementing the java.lang.Runnable interface

public class Runnable implements java.lang.Runnable {  
 @Override  
 public void run() {  
 for(int i =0; i< 100;++i) {  
 System.*out*.println("Thread Class : " + Thread.*currentThread*().getId() + " Value :" + i);  
 }  
 }  
}

The class implements the run() method of Runnable interface. Note that an object of this class is not a thread; it is merely runnable in the sense that its run(), method can be executed concurrently with other codes. The thread object is created using any of the following constructors  
of Thread class :

Thread t = new Thread(new Runnable());  
t.start();

public class CreateThread {  
  
 public void thread1 () {  
 for(int i =0; i < 20;++i) {  
 System.*out*.println(Thread.*currentThread*().getName() + " 111 " + i);  
 }  
 }  
  
 public void thread2 () {  
 for(int i =0; i < 10;++i) {  
 System.*out*.println(Thread.*currentThread*().getName() + " 222 " + i);  
 }  
 }  
  
  
}

public class Test {  
  
 public static void main(String []arg) {  
  
 CreateThread ct = new CreateThread();  
 *startThread*(()->ct.thread1());  
 *startThread*(()->ct.thread2());  
  
 }  
  
 public static void startThread(Runnable r) {  
  
 Thread t = new Thread(r);  
 t.start();  
  
 }  
  
  
}

# Annotations

Creating an annotation requires two pieces of information:

1. a retention policy and (2) a target

A **retention policy** specifies how long, in terms of the program lifecycle, the annotation should be retained for

The **target** of an annotation specifies which Java constructs an annotation can be applied to

|  |  |
| --- | --- |
| **Element Types** | **Where the annotation can be applied** |
| TYPE | class, interface or enumeration |
| FIELD | fields |
| METHOD | methods |
| CONSTRUCTOR | constructors |
| LOCAL\_VARIABLE | local variables |
| ANNOTATION\_TYPE | annotation type |
| PARAMETER | parameter |

import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*FIELD*)  
public @interface JsonField {  
 public String value() default "";  
}

public class Car {  
  
 @JsonField("Model")  
 private final String model;  
  
 @JsonField("Manufacture")  
 private final String make;  
  
  
 public Car(String make, String model) {  
 this.make = make;  
 this.model = model;  
 }  
 public String getMake() {  
 return make;  
 }  
 public String getModel() {  
 return model;  
 }  
}

import java.lang.reflect.Field;  
import java.lang.reflect.Method;  
import java.util.HashMap;  
import java.util.Map;  
  
import static java.util.Objects.*requireNonNull*;  
import static java.util.stream.Collectors.*joining*;  
  
public class JsonSerializer {  
  
 public String serialize(Object object) {  
  
 Class<?> objectClass = *requireNonNull*(object).getClass();  
  
 for(Method method : objectClass.getMethods()) {  
 System.*out*.println("Method : " + method.getName());  
 }  
  
 Map<String, String> jsonElements = new HashMap<>();  
  
 for (Field field: objectClass.getDeclaredFields()) {  
  
 System.*out*.println("Field value : "+ field.getName());  
  
 field.setAccessible(true);  
  
 if (field.isAnnotationPresent(JsonField.class)) {  
  
 try {  
 jsonElements.put(*getSerializedKey*(field), (String) field.get(object));  
 //System.out.println("jsonElement : " + jsonElements);  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }  
 return toJsonString(jsonElements);  
 }  
  
 private String toJsonString(Map<String, String> jsonMap) {  
 String elementsString = jsonMap.entrySet()  
 .stream()  
 .map(entry -> "\"" + entry.getKey() + "\":\"" + entry.getValue() + "\"")  
 .collect(*joining*(","));  
 return "{" + elementsString + "}";  
 }  
  
 private static String getSerializedKey(Field field) {  
 String annotationValue = field.getAnnotation(JsonField.class).value();  
 if (annotationValue.isEmpty()) {  
 return field.getName();  
 }  
 else {  
 return annotationValue;  
 }  
 }  
}

**REFLECTION + RESOURCEHEADER**

import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*TYPE*)  
public @interface ResourceHeader {  
 public String value() default "";  
}

@ResourceHeader("controller")  
public class Car {  
  
  
 public void custom() {  
  
 System.*out*.println("custom of Car class is calling");  
  
 }  
}

import java.lang.annotation.Annotation;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
  
import static java.util.Objects.*requireNonNull*;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 String value ="controller";  
  
 Car car = new Car();  
  
 Class<?> object = car.getClass();  
 Annotation[] ans = object.getAnnotations();  
  
 for(Annotation a : ans) {  
 if(a instanceof ResourceHeader){  
 ResourceHeader r = (ResourceHeader)a;  
 String file = r.value();  
  
 if(file.equals(value)) {  
  
 try {  
 Method method = object.getMethod("custom");  
 method.invoke(car);  
 } catch (NoSuchMethodException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (InvocationTargetException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
 }  
 }  
}

**SERIALIZING OBJECTS**

**The process of storing and retrieving objects in an external fi le is called *serialization.***

**WRITING AN OBJECT TO A FILE**

import java.io.ObjectOutputStream;  
import java.io.OutputStream;  
import java.io.Serializable;  
import java.nio.file.Files;  
  
public class Car implements Serializable {  
  
 private static final long *serialVersionUID* = 9002L;  
 private int model;  
  
 public void custom() {  
  
 System.*out*.println("custom of Car class is calling");  
  
  
 }  
}

public class Test {  
  
 public static void main(String []arg) {  
  
 Car c = new Car();  
  
 Path file = Paths.*get*("d:\\DataFile\\Object.txt");  
  
 try {  
 ObjectOutputStream out = new ObjectOutputStream(Files.*newOutputStream*(file));  
  
 out.writeObject(c);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 }

**READING AN OBJECT FROM A FILE**

import java.io.BufferedInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
  
import static java.util.Objects.*requireNonNull*;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
  
  
 Path file = Paths.*get*("d:\\DataFile\\Object.txt");  
  
 try {  
 ObjectInputStream in = new ObjectInputStream(new BufferedInputStream(  
 Files.*newInputStream*(file)));  
  
 Car obj = (Car)in.readObject();  
 obj.custom();  
  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

**ZIP FOLDER**

import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.zip.ZipEntry;  
import java.util.zip.ZipOutputStream;  
  
public class ZipFolder {  
  
  
 List<String> listFile = new ArrayList<>();  
  
 public void zipFolder(File dir) {  
  
 List<String> files = listAllFile(dir);  
 String zipFolder = dir.getAbsolutePath()+".zip";  
 try {  
  
 FileOutputStream fos = new FileOutputStream(zipFolder);  
 ZipOutputStream zos = new ZipOutputStream(fos);  
  
 for(String filePath : files) {  
  
 System.*out*.println("Zipping : " + filePath);  
  
 String relativePath = filePath.substring(dir.getAbsolutePath().length()+1);  
 ZipEntry ze = new ZipEntry(relativePath);  
 FileInputStream fis = new FileInputStream(filePath);  
 zos.putNextEntry(ze);  
 byte[] buffer = new byte[1024];  
 int len = 0;  
 while((len = fis.read(buffer))>0){  
 zos.write(buffer,0,len);  
 }  
 zos.closeEntry();  
 fis.close();  
  
 }  
  
 zos.close();  
 fos.close();  
  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
  
  
 public List<String> listAllFile(File dir) {  
  
 File[] files = dir.listFiles();  
  
  
 for(File f : files ) {  
 if(f.isFile()) {  
 listFile.add(f.getAbsolutePath());  
  
 } else{  
 listAllFile(f);  
 }  
 }  
  
 return listFile;  
 }  
}

**RUN GROOVY US GROOVYSHELL**

public void process() {

println "WellCome to Groovy Script"

}

import groovy.lang.GroovyShell;  
import groovy.lang.Script;  
  
import java.io.File;  
import java.io.IOException;  
  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 String pathFile = "d:\\Script\\GroovyScript.groovy";  
 File file = new File(pathFile);  
  
 GroovyShell shell = new GroovyShell();  
 try {  
 Script s = shell.parse(file);  
 s.invokeMethod("process",null);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
}

import javax.script.Invocable;  
import javax.script.ScriptEngine;  
import javax.script.ScriptEngineManager;  
import javax.script.ScriptException;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.Reader;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 ScriptEngineManager sem = new ScriptEngineManager();  
 ScriptEngine se = sem.getEngineByName("groovy");  
 String path = "d:\\Script\\GroovyScript.groovy";  
 Reader reader = null;  
 try {  
 reader = new FileReader(path);  
 se.eval(reader);  
  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (ScriptException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

**RUN BAT FILE**

import javax.script.Invocable;  
import javax.script.ScriptEngine;  
import javax.script.ScriptEngineManager;  
import javax.script.ScriptException;  
import java.io.\*;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 try {  
 Process p = Runtime.*getRuntime*().exec("d:\\Script\\ip.bat");  
 InputStream inputstream = p.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(inputstream));  
 String line;  
 while ((line=br.readLine())!=null) {  
 System.*out*.println(line);  
 }  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
  
 }  
}

**JAVA CONTEXT**

The Java Naming and Directory Interface (JNDI) is an API that supports accessing naming and directory services in Java programs

**JMS+ACTIVE MQ**

<dependencies>  
 <dependency>  
 <groupId>javax.jms</groupId>  
 <artifactId>javax.jms-api</artifactId>  
 <version>2.0.1</version>  
 </dependency>

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-all</artifactId>

<version>5.15.7</version>

</dependency>  
</dependencies>

import org.apache.activemq.ActiveMQConnection;  
import org.apache.activemq.ActiveMQConnectionFactory;  
  
import javax.jms.\*;  
  
public class Test {  
  
 //URL of the JMS server. DEFAULT\_BROKER\_URL will just mean that JMS server is on localhost  
 // default broker URL is : ActiveMQConnection.*DEFAULT\_BROKER\_URL =* tcp://localhost:61616"  
  
 private static String *url* = ActiveMQConnection.*DEFAULT\_BROKER\_URL*;  
 private static String *subject* ="JMS-QUEUE";  
  
  
  
 public static void main(String []arg) {  
 *sendMessage*();  
 }  
  
 public static void sendMessage() {  
  
 // Getting JMS connection from the server and starting it  
 ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(*url*);  
 try {  
 Connection connection = connectionFactory.createConnection();  
 connection.start();  
  
 //Creating a non transactional session to send/receive JMS message.  
 Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  
  
 //Destination represents here our queue 'JMS-QUEUE' on the JMS server.  
 //The queue will be created automatically on the server.  
  
 Destination destination = session.createQueue(*subject*);  
  
 // MessageProducer is used for sending messages to the queue.  
 MessageProducer producer = session.createProducer(destination);  
  
 // We will send a small text message  
 TextMessage message = session.createTextMessage("Hello " +*subject*+" Welcome to ActiveMQ.");  
  
 // Here we are sending our message!  
 producer.send(message);  
  
 System.*out*.println("Sending message to queue !");  
  
 connection.close();  
  
  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

import org.apache.activemq.ActiveMQConnection;  
import org.apache.activemq.ActiveMQConnectionFactory;  
  
import javax.jms.\*;  
  
public class Main {  
  
 // URL of the JMS server  
 private static String *url* = ActiveMQConnection.*DEFAULT\_BROKER\_URL*;  
  
 // Name of the queue we will receive messages from  
 private static String *subject* = "JMS-QUEUE";  
  
 public static void main (String []arg) {  
  
 // Getting JMS connection from the server  
 ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(*url*);  
 Connection connection = null;  
 try {  
 connection = connectionFactory.createConnection();  
 connection.start();  
  
 // Creating session for seding messages  
 Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  
  
 // Getting the queue 'JMS-QUEUE'  
 Destination destination = session.createQueue(*subject*);  
  
 // MessageConsumer is used for receiving (consuming) messages  
 MessageConsumer messageConsumer = session.createConsumer(destination);  
  
 // Here we receive the message.  
 Message message = messageConsumer.receive();  
  
 // We will be using TestMessage in our example. MessageProducer sent us a TextMessage  
 // so we must cast to it to get access to its .getText() method.  
  
 if (message instanceof TextMessage) {  
 TextMessage textMessage = (TextMessage) message;  
 System.*out*.println("Received message '" + textMessage.getText() + "'");  
 }  
 connection.close();  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
  
  
  
  
 }  
}

**LIST ALL FILES**

import org.apache.commons.io.FileUtils;  
import org.apache.commons.io.filefilter.TrueFileFilter;  
import java.io.File;  
import java.util.List;  
  
public class Test {  
 public static void main(String []arg) {  
 String rootPath = "d:\\CAPM\_Source\\";  
 *listAllFiles*(rootPath);  
 }  
  
 public static void listAllFiles(String rootPath) {  
 File dir = new File(rootPath);  
 List<File> files = (List<File>) FileUtils.*listFiles*(dir,  
 TrueFileFilter.*INSTANCE*, TrueFileFilter.*INSTANCE*);  
  
 for(File f : files) {  
 if(f.getName().endsWith(".java")) {  
 System.*out*.println(f.getAbsolutePath());  
 }  
 }  
  
 }  
  
}

**READING CONTENTS FROM TEXT FILES ALL**

**AT ONCE**

import org.apache.commons.io.FileUtils;  
import java.io.File;  
import java.io.IOException;  
  
public class Test {  
 public static void main(String []arg) {  
 String rootPath = "d:\\code.txt";  
 *readAllContext*(rootPath);  
 }  
  
 public static void readAllContext(String rootPath) {  
 File file = new File(rootPath);  
 try {  
 String text = FileUtils.*readFileToString*(file, "UTF-8");  
 System.*out*.println(text);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**EXTRACTING PDF TEXT USING APACHE TIKA**

import org.apache.commons.io.FileUtils;  
import org.apache.tika.exception.TikaException;  
import org.apache.tika.metadata.Metadata;  
import org.apache.tika.parser.AutoDetectParser;  
import org.apache.tika.parser.ParseContext;  
import org.apache.tika.sax.BodyContentHandler;  
import org.xml.sax.SAXException;  
  
import java.io.\*;  
  
public class Test {  
 public static void main(String []arg) {  
 String fileName = "d:\\Cisco ACI\\Introduction to SDN.pdf";  
 *convertPDF*(fileName);  
 }  
  
 public static void convertPDF(String fileName) {  
  
 try {  
 InputStream is = new FileInputStream(fileName);  
 AutoDetectParser parser = new AutoDetectParser();  
 BodyContentHandler handler = new BodyContentHandler(-1);  
 Metadata metadata = new Metadata();  
  
 parser.parse(is,handler,metadata,new ParseContext());  
 System.*out*.println(handler.toString());  
 is.close();  
  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (SAXException e) {  
 e.printStackTrace();  
 } catch (TikaException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
}

**WRITE JSON**

import org.json.JSONArray;  
import org.json.JSONObject;  
  
import java.io.FileWriter;  
import java.io.IOException;  
  
  
public class Test {  
 public static void main(String []arg) {  
 String fileName = "d:\\Cisco ACI\\Introduction to SDN.pdf";  
 *writeJson*(fileName);  
 }  
  
 public static void writeJson(String fileName) {  
 JSONObject jsonObject = new JSONObject();  
 jsonObject.put("auhthor","Steve");  
 jsonObject.put("Name", "IT");  
 JSONArray list = new JSONArray();  
  
 list.put("There are characters in this book that will remind us ");  
 list.put("Hogwarts is a truly magical place, not only in the most ");  
 list.put("Parents need to know that this thrill-a-minute story");  
  
 jsonObject.put("Mess", list);  
  
 try {  
 FileWriter fw = new FileWriter("d:\\DataFile\\result.json");  
 fw.write(jsonObject.toString());  
 fw.flush();  
 fw.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println(jsonObject);  
  
  
 }  
}

**READING JSON FILES USING JSON.SIMPLE**

import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;  
import org.json.simple.parser.JSONParser;  
import org.json.simple.parser.ParseException;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.Iterator;  
  
public class Test {  
 public static void main(String []arg) {  
 String fileName = "d:\\DataFile\\Input.js";  
 *parseJson*(fileName);  
 }  
  
 public static void parseJson(String fileName) {  
  
 JSONParser jsonParser = new JSONParser();  
  
 try {  
 Object obj = jsonParser.parse(new FileReader(fileName));  
  
 JSONObject jsonObject = (JSONObject) obj;  
 String metric = (String) jsonObject.get("metric\_id");  
 System.*out*.println(metric);  
  
 JSONArray jsonArray = (JSONArray) jsonObject.get("series");  
 Iterator<JSONObject> iterator = jsonArray.iterator();  
 while (iterator.hasNext()) {  
 JSONObject JSONObjectHeader = (JSONObject) iterator.next().get("header");  
 String name =(String)JSONObjectHeader.get("name");  
 System.*out*.println(name);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

**SQL**

1. CREATE DATABASE DB;
2. DROP DATABASE DB;
3. CREATE TABLE MF (

Name varchar(255),

Description varchar(255),

Value int

);

1. DROP TABLE mf
2. SELECT \*FROM mf
3. INSERT INTO mf (Name,Description,Value) VALUES ('CPU','Cisco',75)

////////////////////////////////////////////////////////////////////////////////////////////

[Config for mysql]

mysql> CREATE USER 'app'@'localhost' IDENTIFIED BY 'app';

mysql> GRANT ALL PRIVILEGES ON \*.\* TO 'app'@'localhost' WITH GRANT OPTION;

mysql> CREATE USER 'app'@'%' IDENTIFIED BY 'app';

mysql> GRANT ALL PRIVILEGES ON \*.\* TO 'app'@'%' WITH GRANT OPTION;

mysql> FLUSH PRIVILEGES;

//////////////////////////////////////////////////////////////////////////////////////////////

import com.mysql.cj.jdbc.MysqlDataSource;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class Test {  
 public static void main(String []arg) {  
 *readDataBase*();  
 }  
  
 public static void readDataBase() {  
 String username ="java";  
 String password ="1234";  
 MysqlDataSource mysqlDataSource = new MysqlDataSource();  
 mysqlDataSource.setURL("jdbc:mysql://192.168.0.100:3306/db");  
 mysqlDataSource.setUser(username);  
 mysqlDataSource.setPassword(password);  
 String cmd = "SELECT \*FROM mf";  
  
 try {  
 Connection conn = mysqlDataSource.getConnection();  
 Statement stateMent = conn.createStatement();  
 ResultSet resultSet = stateMent.executeQuery(cmd);  
  
 while (resultSet.next()) {  
 String name = resultSet.getString("Name");  
 int value = resultSet.getInt("Value");  
 System.*out*.println(name + " " + value);  
 }  
  
 resultSet.close();  
 stateMent.close();  
 conn.close();  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

import com.mysql.cj.jdbc.MysqlDataSource;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class Test {  
 public static void main(String []arg) {  
 *readDataBase*();  
 }  
  
 public static void readDataBase() {  
 String username ="java";  
 String password ="1234";  
 MysqlDataSource mysqlDataSource = new MysqlDataSource();  
 mysqlDataSource.setURL("jdbc:mysql://192.168.0.100:3306/db");  
 mysqlDataSource.setUser(username);  
 mysqlDataSource.setPassword(password);  
 String cmd = "INSERT INTO mf (Name,Description,Value) VALUES ('Disk','Cisco',55)";  
  
 try {  
 Connection conn = mysqlDataSource.getConnection();  
 Statement stateMent = conn.createStatement();  
 stateMent.executeUpdate(cmd);  
  
  
 stateMent.close();  
 conn.close();  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

**LOG S4J**

<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.5</version>  
</dependency>  
<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-log4j12</artifactId>  
 <version>1.7.5</version>  
</dependency>

File : log4j.properties

log4j.rootLogger=DEBUG, STDOUT, file  
  
log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender  
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout  
log4j.appender.STDOUT.layout.ConversionPattern=%d{dd-MM-yyyy HH:mm:ss} %5p [%t] (%F:%L) - %m%n  
  
log4j.appender.file=org.apache.log4j.RollingFileAppender  
log4j.appender.file.File=d:\\gateway.log  
log4j.appender.file.layout=org.apache.log4j.PatternLayout  
log4j.appender.file.layout.ConversionPattern=%d{dd-MM-yyyy HH:mm:ss} %-5p %c{1}:%L - %m%n

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class Test {  
  
 public static final Logger *LOGGER* = LoggerFactory.*getLogger*(Test.class);  
  
 public static void main(String []arg) {  
 *s4jLogger*();  
 }  
  
 public static void s4jLogger() {  
 System.*out*.println (" Start ");  
 *LOGGER*.info("DebugPut : []");  
  
 }  
}

**HIBERNATE**

**MICROSOFT SQL**

<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>  
 <property name="hibernate.connection.driver\_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>  
 <property name="hibernate.connection.url">jdbc:sqlserver://localhost:1433;databaseName=CertProject</property>  
 <property name="hibernate.connection.username">sa</property>  
 <property name="hibernate.connection.password">123456</property>  
 <property name="show\_sql">true</property>  
 <property name="hbm2ddl.auto">update</property>  
 <mapping class="DiskDrive"/>  
 </session-factory>  
</hibernate-configuration>

**MySQL**

<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>  
 <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  
 <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/data</property>  
 <property name="hibernate.connection.username">java</property>  
 <property name="hibernate.connection.password">1234</property>  
 <property name="show\_sql">true</property>  
 <property name="hbm2ddl.auto">update</property>  
 <mapping class="DiskDrive"/>  
 </session-factory>  
</hibernate-configuration>

import javax.persistence.\*;  
  
@Entity  
@Table(name="DiskDrive")  
public class DiskDrive {  
  
 @Id  
 @Column(name="id")  
 @GeneratedValue(strategy=GenerationType.*AUTO*)  
 private int id;  
  
 @Column  
 private String vendor;  
  
 @Column  
 private double value;  
  
 public void setId(int id) {  
 this.id = id;  
 }  
 public int getId() {  
 return id;  
 }  
 public void setValue(double value) {  
 this.value = value;  
 }  
 public double getValue() {  
 return value;  
 }  
  
 public void setVendor(String vendor) {  
 this.vendor = vendor;  
 }  
  
 public String getVendor() {  
 return vendor;  
 }  
}

@Entity annotation specifies that the bean we have written is an entity.

@Table annotation maps the entity to the relational table of a database. If the @Table is not specified, the entity will be mapped with a relational table represented by the simple name of the class.

@Id annotation that declares a field as primary key.

@Column annotation maps the Java field to the field name of the relational table.

@GeneratedValue specifies that the key is automatically generated.

import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
  
public class MainHibernate {  
 public static void main(String []arg) {  
 System.*out*.println("Start");  
  
 Configuration configuration = new Configuration();  
 configuration = configuration.configure("hibernate.cfg");  
 SessionFactory sessionFactory = configuration.buildSessionFactory();  
  
 Session session = sessionFactory.openSession();  
 session.getTransaction().begin();  
  
 DiskDrive diskDrive = new DiskDrive();  
 diskDrive.setVendor("Juniper");  
 diskDrive.setValue(1024);  
  
 session.save(diskDrive);  
 session.getTransaction().commit();  
 session.close();  
  
 System.*out*.println("Finish");  
  
  
 }  
}

**HERBINATE >v5**

import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.boot.Metadata;  
import org.hibernate.boot.MetadataSources;  
import org.hibernate.boot.registry.StandardServiceRegistry;  
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;  
  
  
public class MainHibernate {  
 public static void main(String []arg) {  
 System.*out*.println("Start");  
  
  
 StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder().configure("hibernate.cfg").build();  
 Metadata metadata = new MetadataSources(standardRegistry).getMetadataBuilder().build();  
 SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();  
  
 Session session = sessionFactory.openSession();  
 session.getTransaction().begin();  
  
 DiskDrive diskDrive = new DiskDrive();  
 diskDrive.setVendor("Cisco");  
 diskDrive.setValue(1024);  
  
 session.save(diskDrive);  
 session.getTransaction().commit();  
 session.close();  
  
 System.*out*.println("Finish");  
  
  
 }  
}

**JMS CONTEXT**

import javax.jms.\*;  
import javax.naming.Context;  
import javax.naming.InitialContext;  
import javax.naming.NamingException;  
import java.util.Properties;  
  
public class MainHibernate {  
 public static void main(String []arg) {  
 System.*out*.println("Start");  
  
  
 *senMessageJMS*();  
  
  
  
 System.*out*.println("Finish");  
  
  
 }  
  
 public static void senMessageJMS() {  
  
  
 int port = 61616;  
 String subject ="java:JMS\_QUEUE";  
  
 Properties env = new Properties();  
  
 env.put(Context.*INITIAL\_CONTEXT\_FACTORY*,"org.apache.activemq.jndi.ActiveMQInitialContextFactory");  
 env.put(Context.*PROVIDER\_URL*,"tcp://localhost:61616");  
 env.put("topic.MyTopic",subject);  
  
 try {  
  
 Context context = new InitialContext(env);  
  
 ConnectionFactory connectionFactory = (ConnectionFactory) context  
 .lookup("ConnectionFactory");  
 Connection connection =connectionFactory.createConnection();  
 connection.start();  
  
 Session session = connection.createSession(false, Session.*AUTO\_ACKNOWLEDGE*);  
  
  
 Destination destination = session.createQueue(subject);  
  
  
 MessageProducer producer = session.createProducer(destination);  
  
  
 TextMessage message = session.createTextMessage("Hello " +subject+" Welcome to ActiveMQ.");  
  
  
 producer.send(message);  
  
 System.*out*.println("Sending message to queue !");  
  
 connection.close();  
  
 } catch (NamingException e) {  
 e.printStackTrace();  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
  
}

import javax.jms.\*;  
import javax.naming.Context;  
import javax.naming.InitialContext;  
import javax.naming.NamingException;  
import java.util.Properties;  
  
public class MainHibernate {  
 public static void main(String []arg) {  
 System.*out*.println("Start");  
 *senMessageJMS*();  
 System.*out*.println("Finish");  
 }  
  
 public static void senMessageJMS() {  
 int port = 61616;  
 String subject ="java:JMS\_QUEUE";  
 Properties env = new Properties();  
 env.put(Context.*INITIAL\_CONTEXT\_FACTORY*,"org.apache.activemq.jndi.ActiveMQInitialContextFactory");  
 env.put(Context.*PROVIDER\_URL*,"tcp://localhost:61616");  
 env.put("queue.Queue", "JMS\_QUEUE");  
  
 try {  
  
 Context context = new InitialContext(env);  
  
 QueueConnectionFactory connectionFactory = (QueueConnectionFactory) context  
 .lookup("ConnectionFactory");  
  
 QueueConnection connection =connectionFactory.createQueueConnection();  
  
  
 QueueSession session = connection.createQueueSession(false, Session.*AUTO\_ACKNOWLEDGE*);  
  
 Queue queue = (Queue) context.lookup("Queue");  
  
 QueueSender sender = session.createSender(queue);  
 connection.start();  
 TextMessage message = session.createTextMessage("Hello " +subject+" Welcome to ActiveMQ.");  
  
 sender.send(message);  
  
 connection.close();  
  
  
 } catch (NamingException e) {  
 e.printStackTrace();  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
  
}

**COVERT FILE TO STRING**

<dependency>  
 <groupId>commons-io</groupId>  
 <artifactId>commons-io</artifactId>  
 <version>2.6</version>  
</dependency>

public static void convertFile2String(){  
 String path = "d:\\gateway.log";  
 File file = new File(path);  
 URI uri =file.toURI();  
 try {  
 String message = IOUtils.*toString*(uri,"UTF-8");  
 System.*out*.println(message);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**MAVEN JMS-SQL-HERBINATE**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>groupId</groupId>  
 <artifactId>HibernateProject</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <dependencies>  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-agroal</artifactId>  
 <version>5.3.7.Final</version>  
 <type>pom</type>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.12</version>  
 </dependency>  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-core</artifactId>  
 <version>5.3.7.Final</version>  
 </dependency>  
 <dependency>  
 <groupId>javax.jms</groupId>  
 <artifactId>javax.jms-api</artifactId>  
 <version>2.0.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>activemq-all</artifactId>  
 <version>5.15.7</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>activemq-core</artifactId>  
 <version>5.2.0</version>  
 </dependency>  
 <dependency>  
 <groupId>commons-io</groupId>  
 <artifactId>commons-io</artifactId>  
 <version>2.6</version>  
 </dependency>  
 <dependency>  
 <groupId>com.microsoft.sqlserver</groupId>  
 <artifactId>sqljdbc4</artifactId>  
 <version>4.0</version>  
 </dependency>  
  
 </dependencies>  
   
</project>

**WILDFLY CONFIGURATION**

If, on the other hand, you want to bind the network interface to all available sets of IP addresses, you can use the <any-address /> element, as follows:

For example :

<interfaces>  
 <interface name="management">  
 <any-address/>  
 </interface>  
 <interface name="public">  
 <any-address/>  
 </interface>  
 <interface name="unsecure">  
 <any-address/>  
 </interface>  
</interfaces>

**The socket-binding groups :**

A socket-binding group defines a logical name for a socket

**Configuring application server logging**

The full range of log levels, from lowest to highest, are: OFF, FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE, and ALL

**The size-rotating-file-handler**

The size-rotating-file-handler defines a handler that writes to a file, rotating the log after the size of the file grows beyond a certain point. It also keeps a fixed number of backups.

For example :

<size-rotating-file-handler name="GATEWAY\_LOGGER" autoflush="true">  
 <formatter>  
 <named-formatter name="GATEWAY\_LOGGING\_PATTERN"/>  
 </formatter>  
 <file relative-to="jboss.server.log.dir" path="gateway.log"/>  
 <rotate-size value="50m"/>  
 <max-backup-index value="7"/>  
 <append value="true"/>  
</size-rotating-file-handler>

**Installing the JDBC driver**

The first step to install a new module is to create the directory structure under the modules folder. The actual path for the module is JBOSS\_HOME/modules/<module>/main.

The main folder is where all the key module components are installed, namely, the driver and the module.xml file. So, next, we need to add the following units:  
JBOSS\_HOME/modules/com/mysql/main/mysql-connector-java-5.1.30-bin.jar  
JBOSS\_HOME/modules/com/mysql/main/module.xml  
The MySQL JDBC driver used in this example, also known as Connector/J, can be downloaded for free from the MySQL site (http://dev.mysql.com/downloads/connector/j/).  
At the time of writing, the latest version is 5.1.30.  
The last thing to do is to create the module.xml file. This file contains the actual module definition. It is important to make sure that the module name (com.mysql) corresponds to the module attribute defined in the your datasource. You must also state the path to the JDBC driver resource and finally add the module dependencies, as shown in the following code:

<module xmlns="urn:jboss:module:1.3" name="com.mysql">   
 <resources>  
 <resource-root path="mysql-connector-java-5.1.30-bin.jar"/>  
 </resources>  
 <dependencies>  
 <module name="javax.api"/>  
 <module name="javax.transaction.api"/>  
 </dependencies>  
</module>

**Adding a local datasource**

<datasource jta="true" jndi-name="java:jboss/datasources/VNA\_MySQL" pool-name="VNA\_MySQL" enabled="true" use-java-context="true" use-ccm="true">  
<connection-url>jdbc:mysql://localhost:3306/vna\_huydu02\_E20570?serverTimezone=America/New\_York&amp;useSSL=false&amp;allowPublicKeyRetrieval=true</connection-url>  
<driver>mysql</driver>  
<security>  
 <security-domain>VNA\_MySQL</security-domain>  
</security>  
</datasource>

<drivers>  
<driver name="h2" module="com.h2database.h2">  
 <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>  
</driver>  
<driver name="mysql" module="com.mysql">  
 <xa-datasource-class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-class>  
</driver>  
</drivers>

You should use an xa-datasource in cases where a single transaction spans multiple datasources, for example, if a method consumes a **Java Message Service** (**JMS**) and updates a **Java Persistence API** (**JPA**) entity

**Configuring the EJB components**

**Configuring the messaging system**

Messaging systems normally support two main styles of asynchronous messaging:  
**Queues** (point-to-point messaging) and **Topics** (publish/subscribe messaging).  
In the point-to-point model, a sender posts messages to a particular queue, and a receiver reads messages from the queue. Here, the sender knows the destination of the message and  
posts the message directly to the receiver’s queue.  
The publish/subscribe model supports the publishing of messages to a particular message topic. Subscribers may register interest in receiving messages on a particular message  
topic. In this model, neither the publisher nor the subscriber know about each other.

**WILDFLY CMD**

**Add New SUBSYSTEM WILDFLY : (Standalone.xml)**

<subsystem xmlns="urn:jboss:domain:messaging-activemq:3.0">  
<server name="default">  
 <security-setting name="#">  
 <role name="guest" send="true" consume="true" create-non-durable-queue="true" delete-non-durable-queue="true"/>  
 </security-setting>  
 <address-setting name="#" dead-letter-address="jms.queue.DLQ" expiry-address="jms.queue.ExpiryQueue" max-size-bytes="10485760" page-size-bytes="2097152" message-counter-history-day-limit="10"/>  
 <http-connector name="http-connector" socket-binding="http" endpoint="http-acceptor"/>  
 <http-connector name="http-connector-throughput" socket-binding="http" endpoint="http-acceptor-throughput">  
 <param name="batch-delay" value="50"/>  
 </http-connector>  
 <in-vm-connector name="in-vm" server-id="0">  
 <param name="buffer-pooling" value="false"/>  
 </in-vm-connector>  
 <http-acceptor name="http-acceptor" http-listener="default"/>  
 <http-acceptor name="http-acceptor-throughput" http-listener="default">  
 <param name="batch-delay" value="50"/>  
 <param name="direct-deliver" value="false"/>  
 </http-acceptor>  
 <in-vm-acceptor name="in-vm" server-id="0">  
 <param name="buffer-pooling" value="false"/>  
 </in-vm-acceptor>  
 <jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>  
 <jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>  
 <jms-queue name="InventoryUpdateQueue" entries="jms/queue/InventoryUpdateQueue java:jboss/exported/jms/queue/InventoryUpdateQueue"/>  
 <jms-queue name="NormalizedInventoryResponseQueue" entries="jms/queue/NormalizedInventoryResponseQueue"/>  
 <jms-queue name="NormalizedPerformanceResponseQueue" entries="jms/queue/NormalizedPerformanceResponseQueue"/>  
 <jms-queue name="NormalizedNotificationResponseQueue" entries="jms/queue/NormalizedNotificationResponseQueue"/>  
 <jms-queue name="SouthboundInventoryResponseQueue" entries="jms/queue/SouthboundInventoryResponseQueue"/>  
 <jms-queue name="OCEngineUpdateQueue" entries="jms/queue/OCEngineUpdateQueue java:jboss/exported/jms/queue/OCEngineUpdateQueue"/>  
 <jms-topic name="ClientUpdateTopic" entries="jms/topic/ClientUpdateTopic"/>  
 <connection-factory name="InVmConnectionFactory" entries="java:/ConnectionFactory" connectors="in-vm"/>  
 <connection-factory name="RemoteConnectionFactory" entries="java:jboss/exported/jms/RemoteConnectionFactory" connectors="http-connector"/>  
 <pooled-connection-factory name="activemq-ra" entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory" connectors="in-vm" transaction="xa"/>  
</server>  
</subsystem>

**SEND MESSAGE VIA WILDFLY**

package main.java;  
  
  
import org.wildfly.naming.client.WildFlyInitialContext;  
  
import javax.jms.\*;  
import javax.naming.Context;  
import javax.naming.NamingException;  
import java.util.Properties;  
  
public class MainHibernate {  
  
 public static void main (String []arg) {  
 System.*out*.println("Start ");  
  
 Properties p = new Properties();  
  
 p.put(Context.*INITIAL\_CONTEXT\_FACTORY*,"org.wildfly.naming.client.WildFlyInitialContextFactory");  
 p.put(Context.*PROVIDER\_URL*, "remote+http://localhost:8080");  
  
  
 try {  
 Context context = new WildFlyInitialContext(p);  
 QueueConnectionFactory queueFactory = (QueueConnectionFactory)context.lookup("jms/RemoteConnectionFactory");  
 try {  
 QueueConnection conn = queueFactory.createQueueConnection("data","data");  
 QueueSession session = conn.createQueueSession( false, Session.*AUTO\_ACKNOWLEDGE* );  
  
 Queue queue = (Queue)context.lookup( "jms/queue/InventoryUpdateQueue");  
  
 QueueSender sender = session.createSender( queue );  
  
 conn.start();  
  
 String message = "Hello Wildfly";  
 TextMessage txtMessage = session.createTextMessage( message );  
 sender.send(txtMessage);  
  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
  
 } catch (NamingException e) {  
 e.printStackTrace();  
 }  
  
  
 System.*out*.println("Message is sent successfully");  
 }  
  
}

**RECEIVE MESSAGE VIA WILDFLY**

package main.java;  
  
  
import org.wildfly.naming.client.WildFlyInitialContext;  
  
import javax.jms.\*;  
import javax.naming.Context;  
import javax.naming.NamingException;  
import java.util.Properties;  
  
public class MainHibernate {  
  
 public static void main (String []arg) {  
 System.out.println("Start ");  
  
 Properties p = new Properties();  
  
 p.put(Context.INITIAL\_CONTEXT\_FACTORY,"org.wildfly.naming.client.WildFlyInitialContextFactory");  
 p.put(Context.PROVIDER\_URL, "remote+http://localhost:8080");  
  
  
 try {  
 Context context = new WildFlyInitialContext(p);  
 QueueConnectionFactory queueFactory = (QueueConnectionFactory)context.lookup("jms/RemoteConnectionFactory");  
 try {  
 QueueConnection conn = queueFactory.createQueueConnection("data","data");  
 QueueSession session = conn.createQueueSession( false, Session.AUTO\_ACKNOWLEDGE );  
  
 Queue queue = (Queue)context.lookup( "jms/queue/InventoryUpdateQueue");  
  
  
 QueueReceiver queueReceiver = session.createReceiver(queue);  
  
 conn.start();  
  
 TextMessage message = (TextMessage) queueReceiver.receive();  
  
 System.out.println("RECEIVED : " + message.getText());  
  
  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
  
 } catch (NamingException e) {  
 e.printStackTrace();  
 }  
  
  
 System.out.println("Message is received successfully");  
 }  
  
}

**COPY FOLDER USE CAMEL**

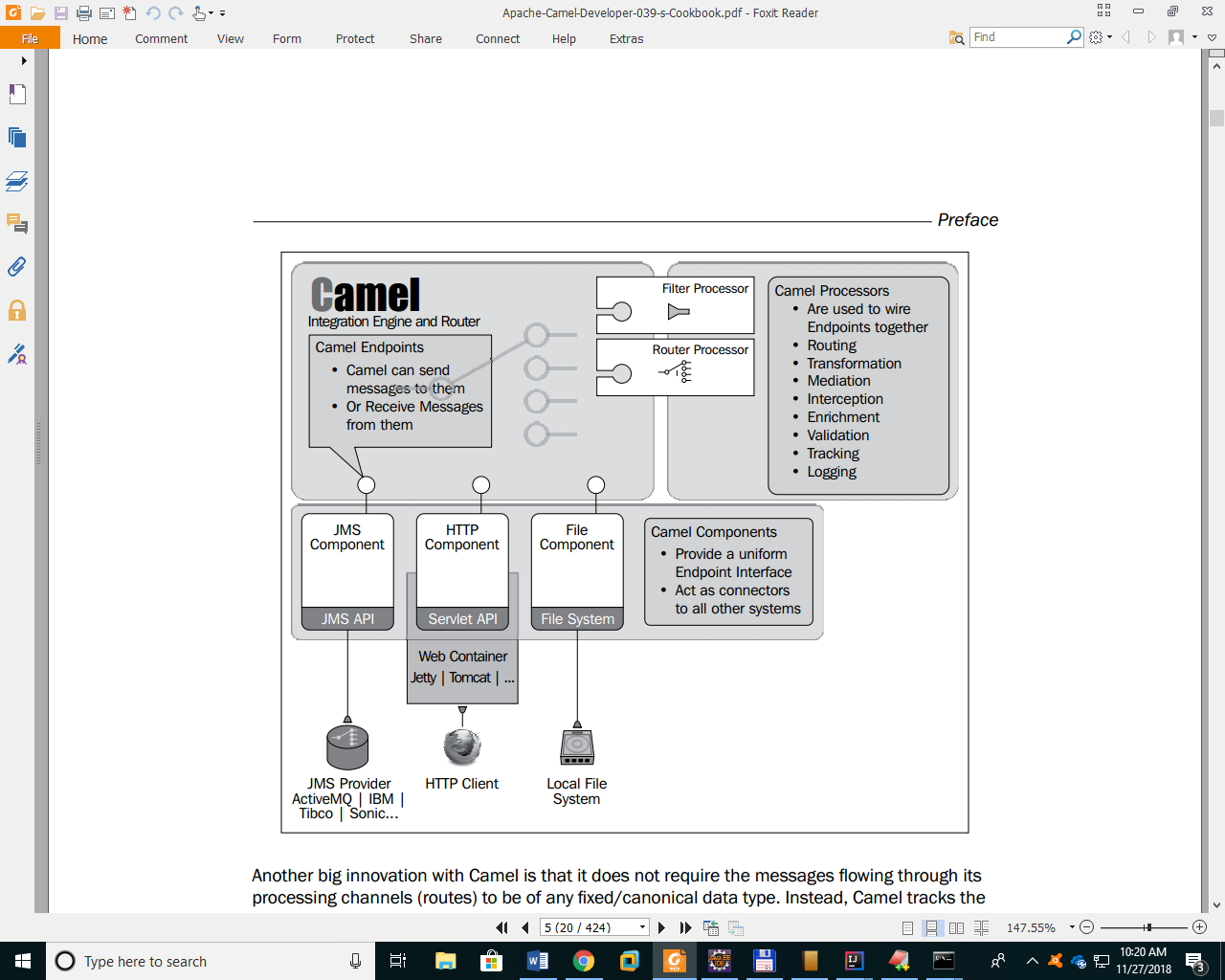
<dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-core</artifactId>  
 <version>2.22.1</version>  
</dependency>

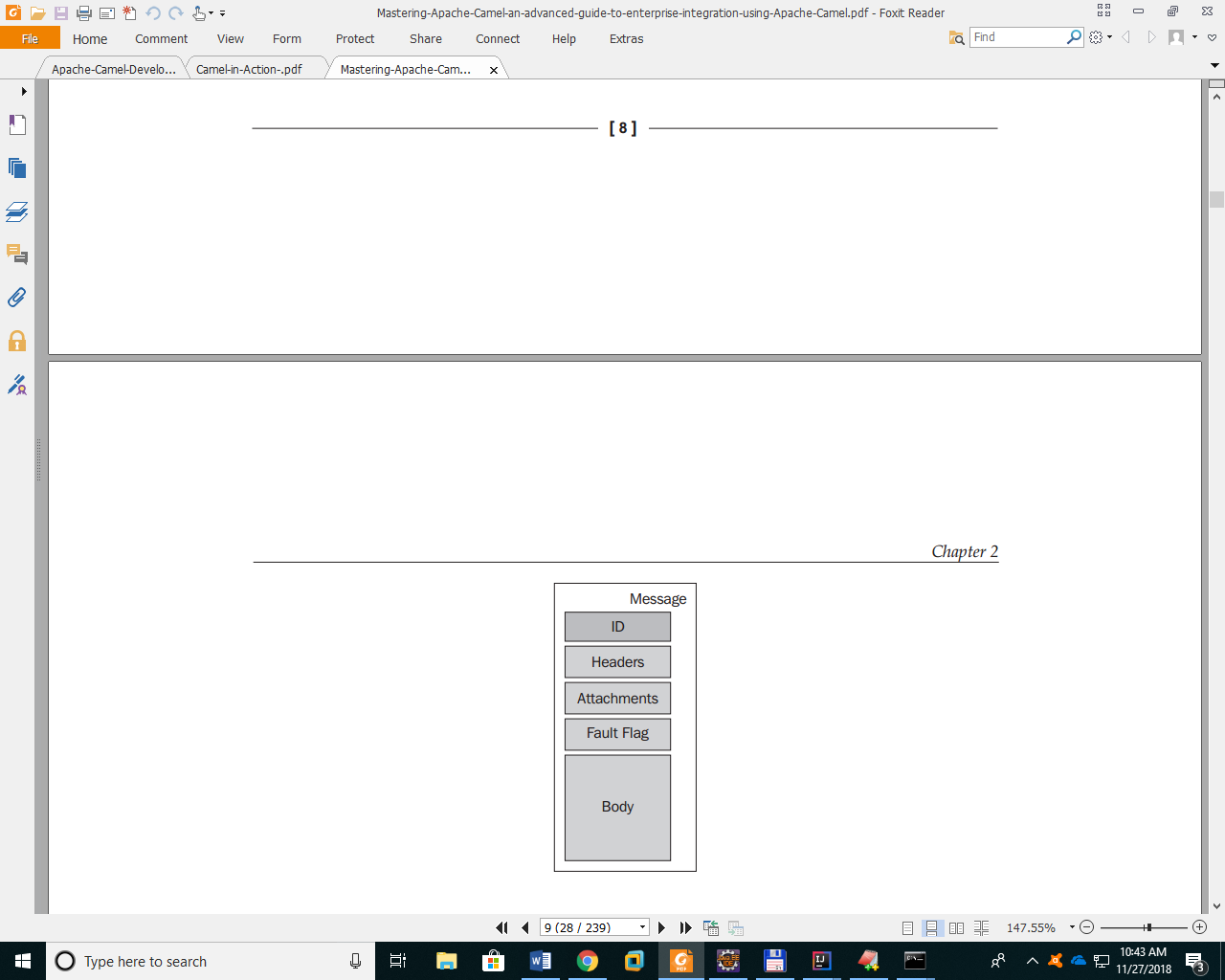
import org.apache.camel.CamelContext;  
import org.apache.camel.builder.RouteBuilder;  
import org.apache.camel.impl.DefaultCamelContext;  
  
public class MainCamel {  
  
 public static void main(String []arg) {  
 System.*out*.println("Start test camel");  
 CamelContext context = new DefaultCamelContext();  
  
 try {  
 context.addRoutes(new RouteBuilder() {  
 public void configure() {  
 from("file:d:\\IELTS\\From Mike\\?noop=true")  
 .to("file:d:\\OutputData\\");  
 }  
 });  
  
 context.start();  
 Thread.*sleep*(10000);  
 context.stop();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
  
 System.*out*.println("End test camel");  
 }  
}

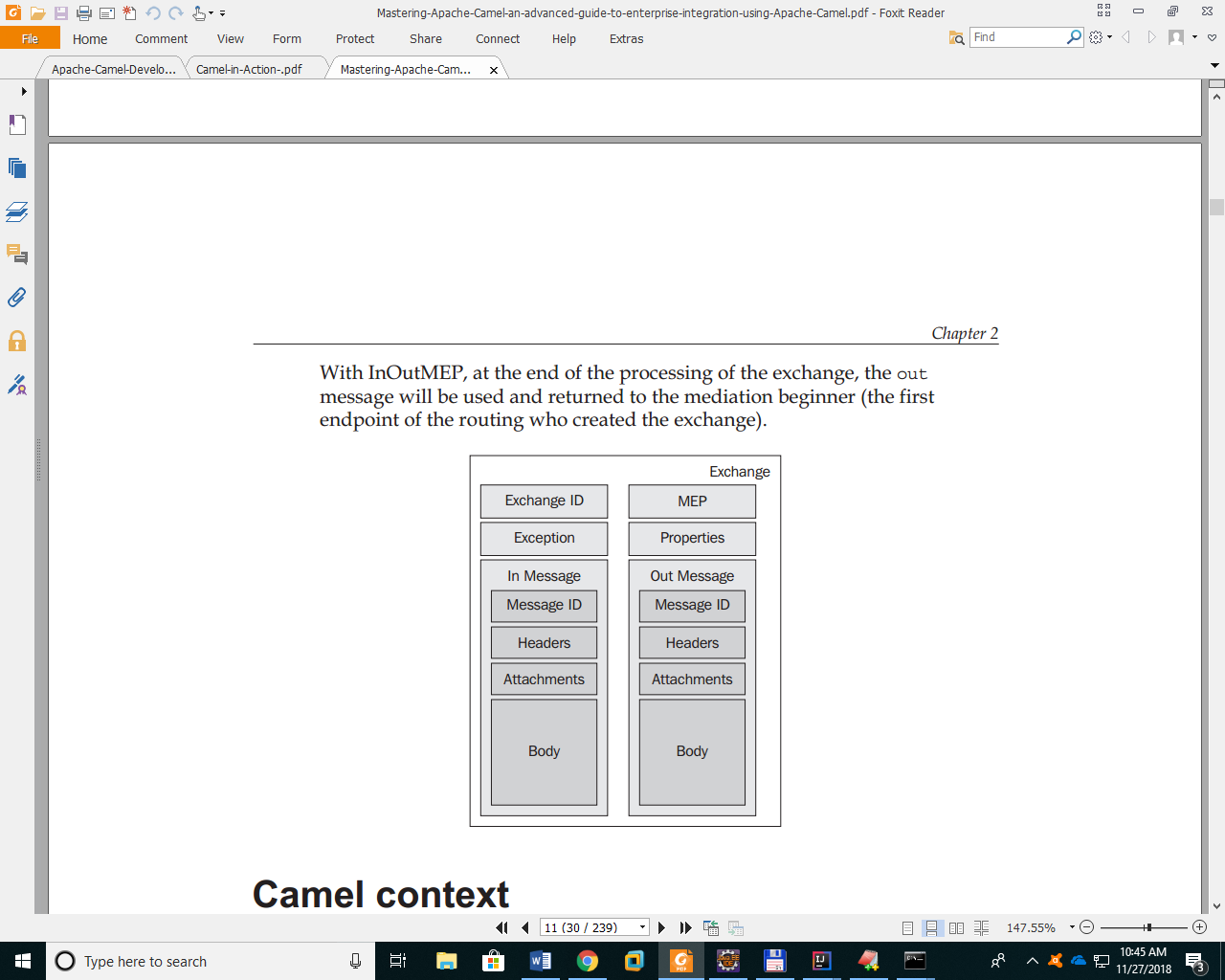
import org.apache.camel.builder.RouteBuilder;  
  
public class CopyFile extends RouteBuilder {  
  
 public void configure() throws Exception {  
 from("file:d:\\Compiler\\Came\\?noop=true")  
 .to("file:d:\\OutputData\\");  
 }  
}

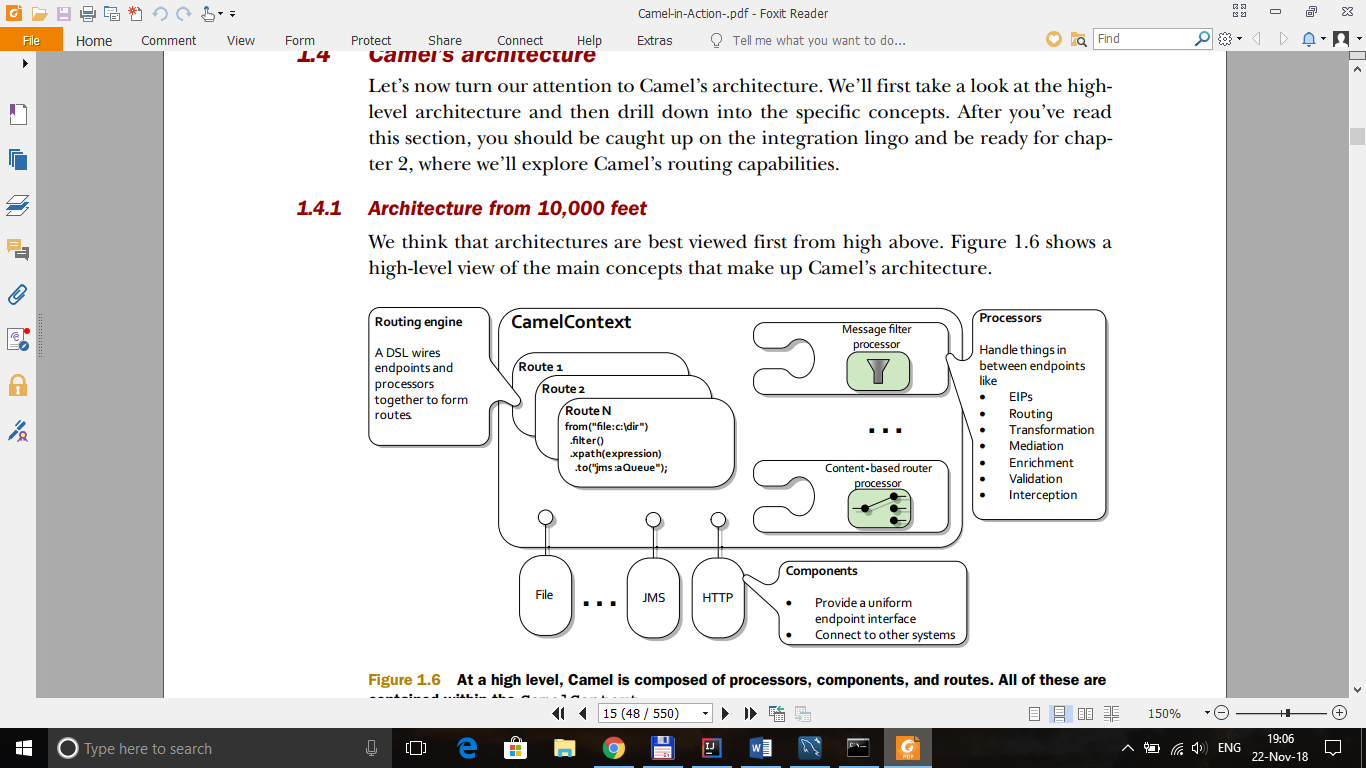
import org.apache.camel.CamelContext;  
import org.apache.camel.builder.RouteBuilder;  
import org.apache.camel.impl.DefaultCamelContext;  
  
public class MainCamel {  
  
 public static void main(String []arg) {  
 System.*out*.println("Start test camel");  
 CamelContext context = new DefaultCamelContext();  
 RouteBuilder r = new CopyFile();  
  
 try {  
 context.addRoutes(r);  
 context.start();  
 Thread.*sleep*(10000);  
 context.stop();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
  
  
  
 System.*out*.println("End test camel");  
 }  
}

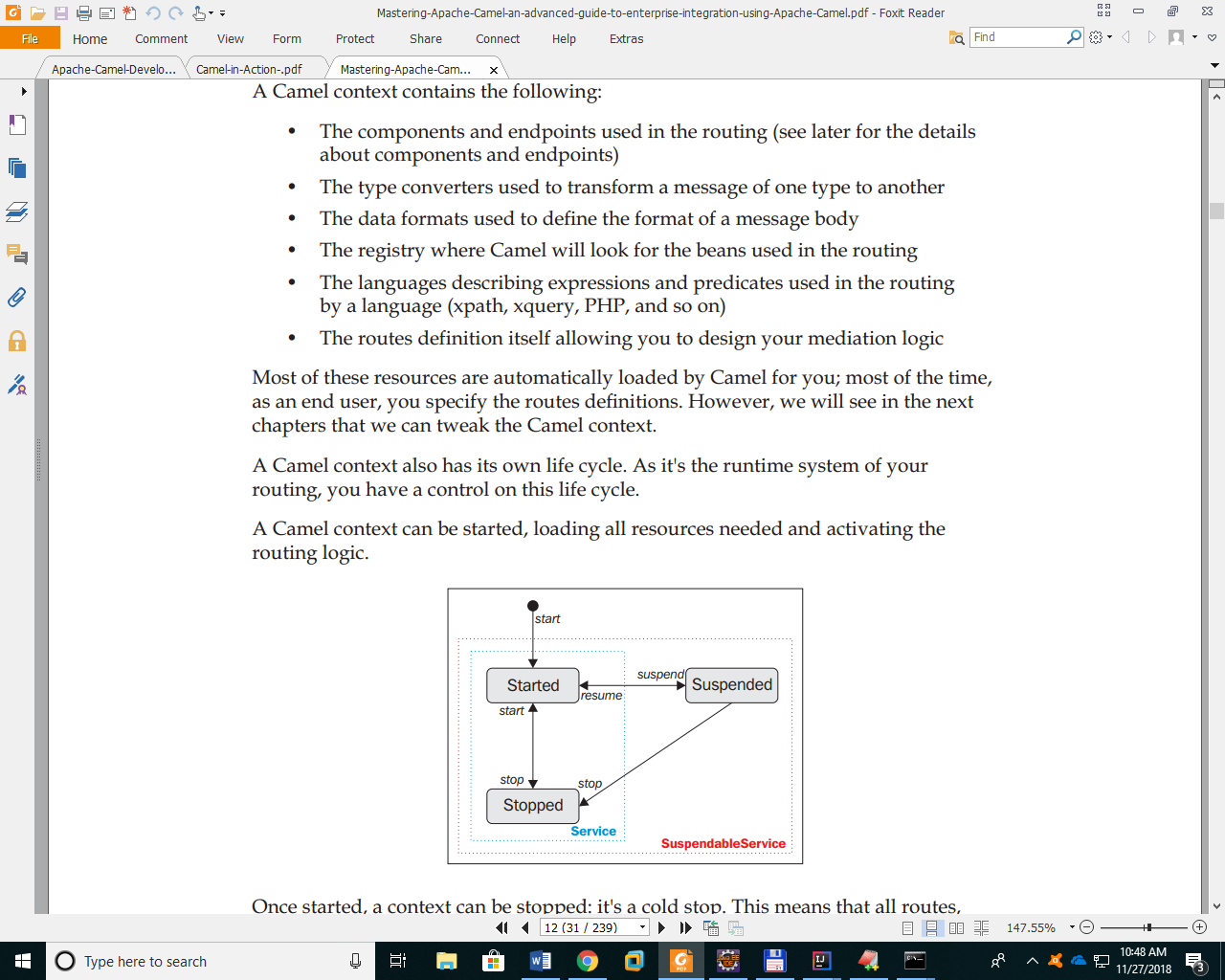
**APACHE CAMEL JMS**



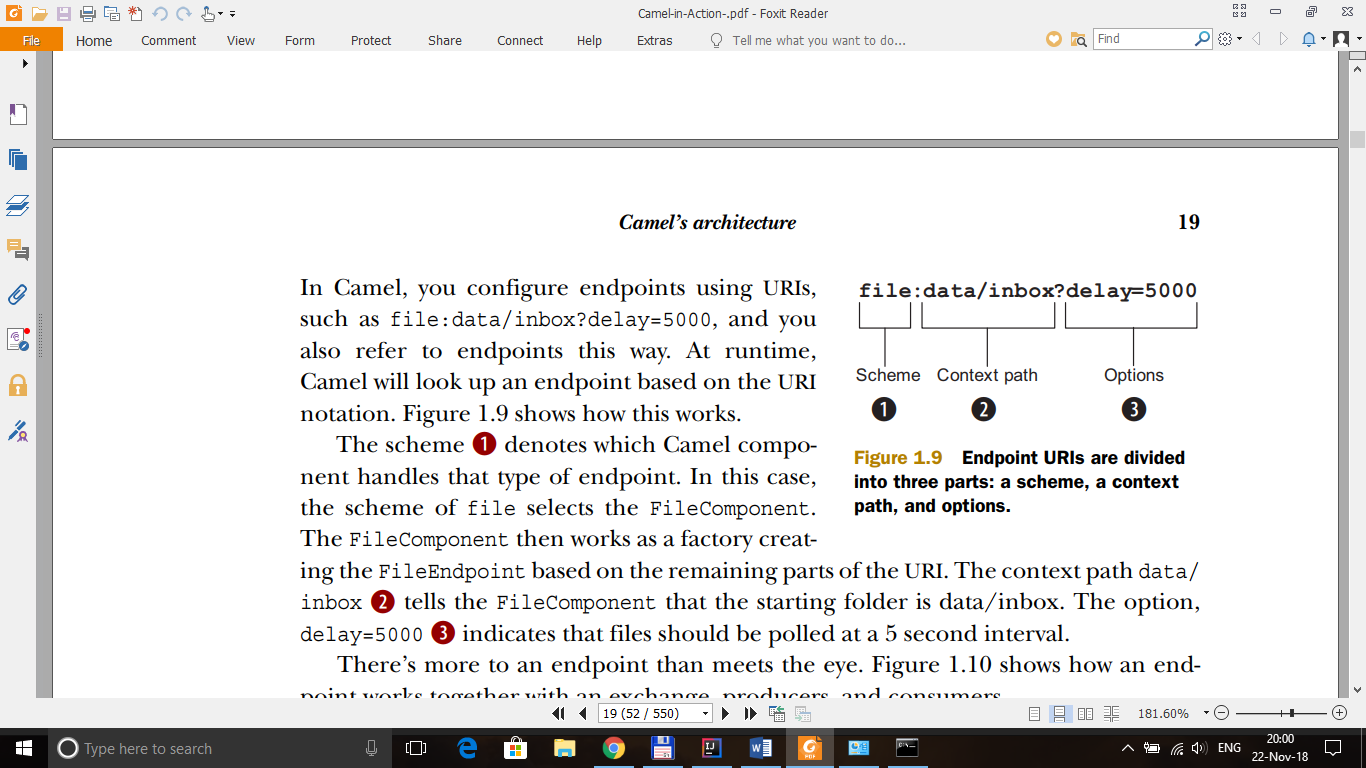




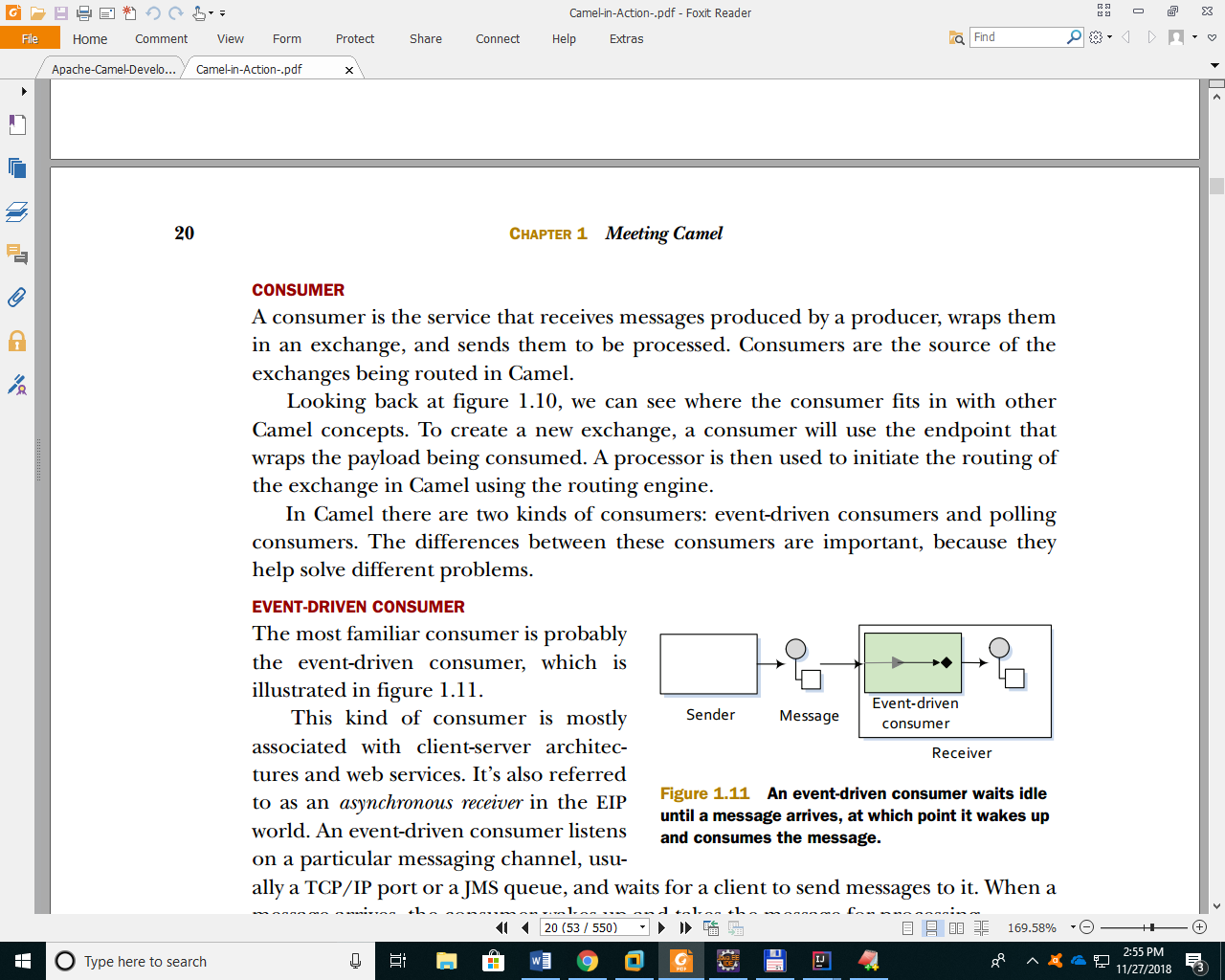




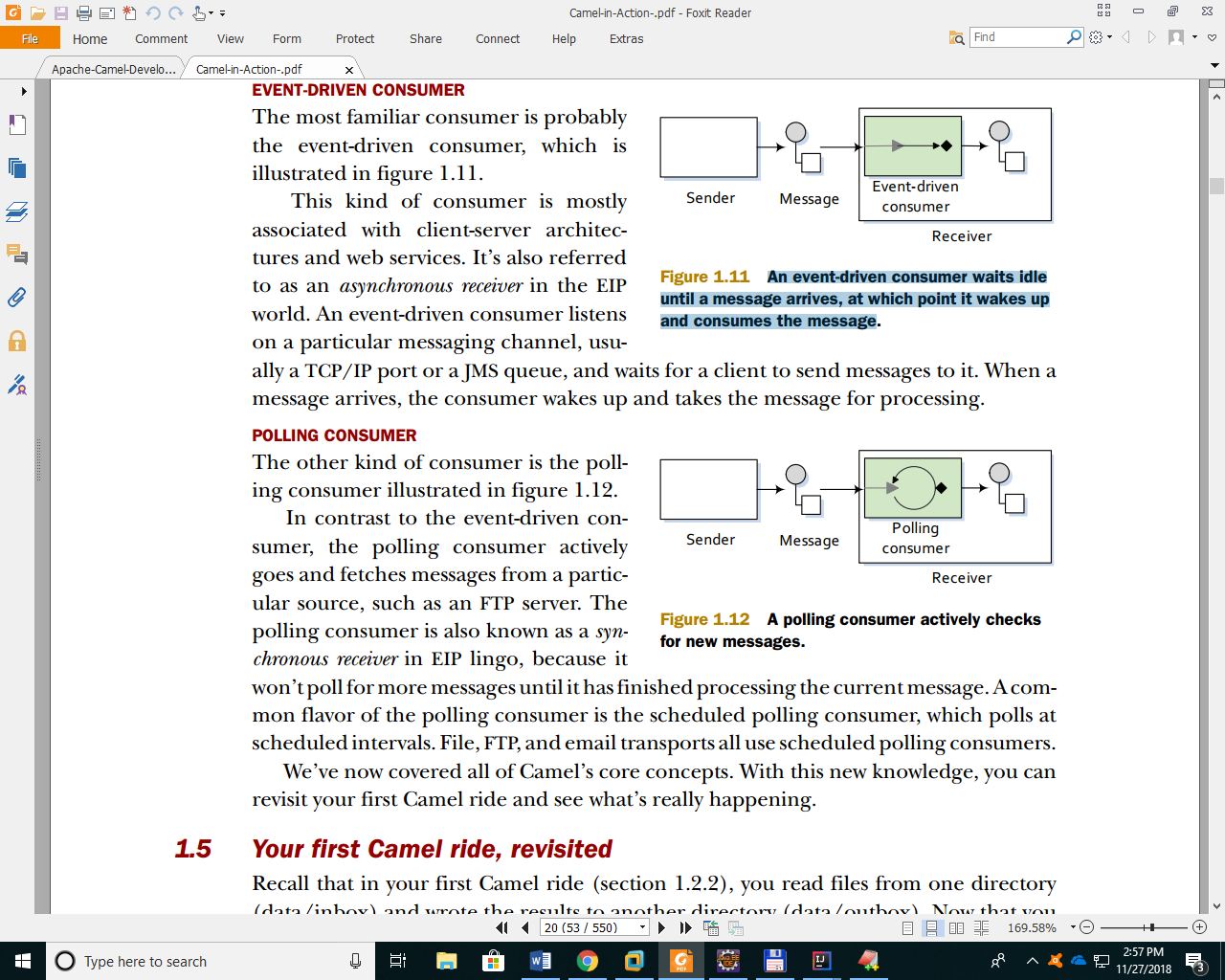
The endpoints are described using a URI in the following format:  
***component:option?option=value&option=value***  
For instance, we can defne an endpoint of the fle component using the  
following code:  
***file:data/inbox?delay=5000&noop=true***

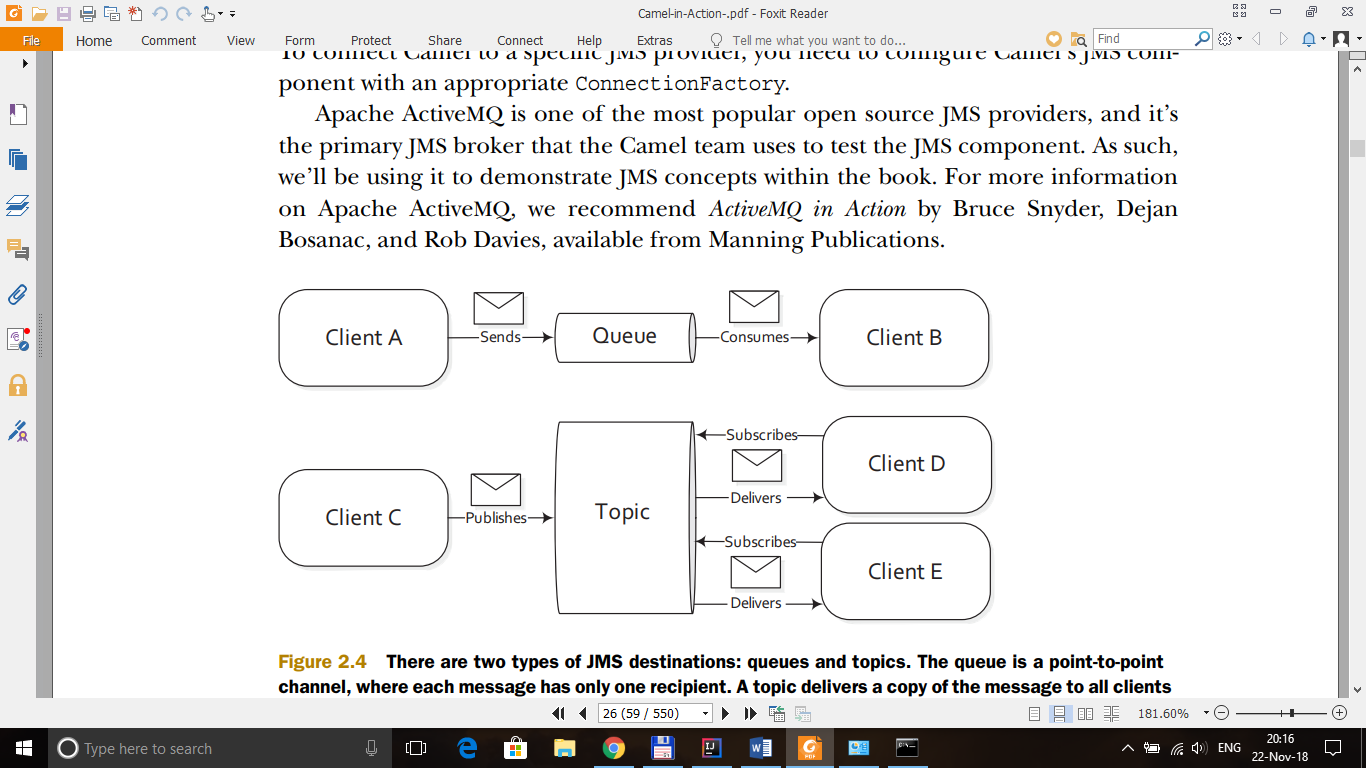


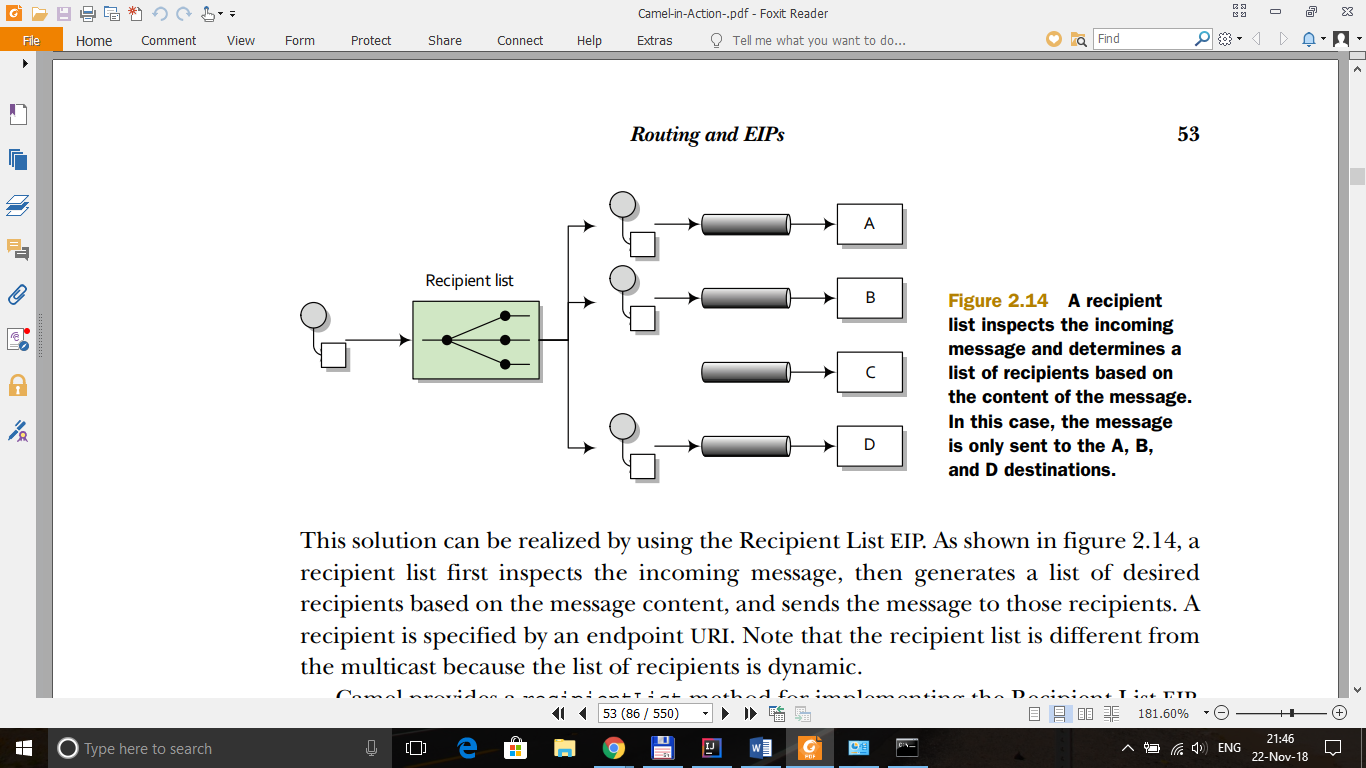
An event-driven consumer waits idle until a message arrives, at which point it wakes up and consumes the message.



A polling consumer actively checks for new messages







If you neglect to call CamelContext.start() in your application then messages will not be processed because internal threads will not have been created.

The Camel Exchange defines two methods for retrieving messages: getIn and getOut. The getIn method returns the incoming message, and the getOut method accesses the outbound message

<dependencies>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-core</artifactId>  
 <version>2.22.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.camel</groupId>  
 <artifactId>camel-jms</artifactId>  
 <version>2.22.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>activemq-all</artifactId>  
 <version>5.15.8</version>  
 </dependency>  
 <dependency>  
 <groupId>javax.jms</groupId>  
 <artifactId>javax.jms-api</artifactId>  
 <version>RELEASE</version>  
 <scope>compile</scope>  
 </dependency>  
</dependencies>

import org.apache.activemq.ActiveMQConnectionFactory;  
import org.apache.camel.CamelContext;  
import org.apache.camel.ProducerTemplate;  
import org.apache.camel.builder.RouteBuilder;  
import org.apache.camel.component.jms.JmsComponent;  
import org.apache.camel.impl.DefaultCamelContext;  
  
import javax.jms.ConnectionFactory;  
  
public class MainCamel {  
  
 public static void main(String []arg) {  
 System.*out*.println("Start test camel");  
 ConnectionFactory connectionFactory =  
 new ActiveMQConnectionFactory("tcp://0.0.0.0:61616");  
 CamelContext context = new DefaultCamelContext();  
 context.addComponent("jms",  
 JmsComponent.*jmsComponentAutoAcknowledge*(connectionFactory));  
  
 RouteBuilder r = new CopyFile();  
  
 try {  
 context.addRoutes(r);  
 ProducerTemplate template = context.createProducerTemplate();  
 context.start();  
 Thread.*sleep*(1000000000);  
 context.stop();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
  
  
  
 System.*out*.println("End test camel");  
 }  
}

import org.apache.camel.builder.RouteBuilder;  
  
public class CopyFile extends RouteBuilder {  
 public void configure() throws Exception {  
 public void configure() throws Exception {  
 //from ("file:d:\\DataFile\\?noop=true").to("activemq:queue:JMS\_QUEUE");  
 from ("activemq:queue:JMS\_QUEUE").to("file:d:\\OutputData\\?noop=true");  
 }  
}

The noop option tells Camel to leave the source file as is

*Queues* are strictly point-to-point, where each message has only one consumer. *Topics* operate on a publish/subscribe scheme; a single message may be delivered to many consumers if they have subscribed to the topic

vm://localhost URI means that you should connect to an embedded broker named “localhost” running inside the current JVM.

Content-Based Router (CBR) is a message router that routes a message to a destination based on its content. The CBR routes messages based on their content. In this case, the filename  
extension (as a message header) is used to determine which queue to route to.

System.*out*.println("Processor : " + exchange.getIn().getHeader("CamelFileName"));

**🡪Print FILE NAME**

**ADD MYSQL TO DATASOURCE WILDFLY**

1. Create module.xml (e:\Software\wildfly-14.0.1.Final\modules\system\layers\base\com\mysql\main\ )

<?xml version="1.0" encoding="UTF-8"?>  
<module name="com.mysql" xmlns="urn:jboss:module:1.5">  
<resources>  
 <resource-root path="mysql-connector-java-5.1.47-bin.jar"/>  
</resources>  
<dependencies>  
 <module name="javax.api"/>  
 <module name="javax.transaction.api"/>  
</dependencies>  
</module>

2. Copy mysql-connector-java-5.1.47-bin.jar to folder \modules\system\layers\base\com\mysql\main\

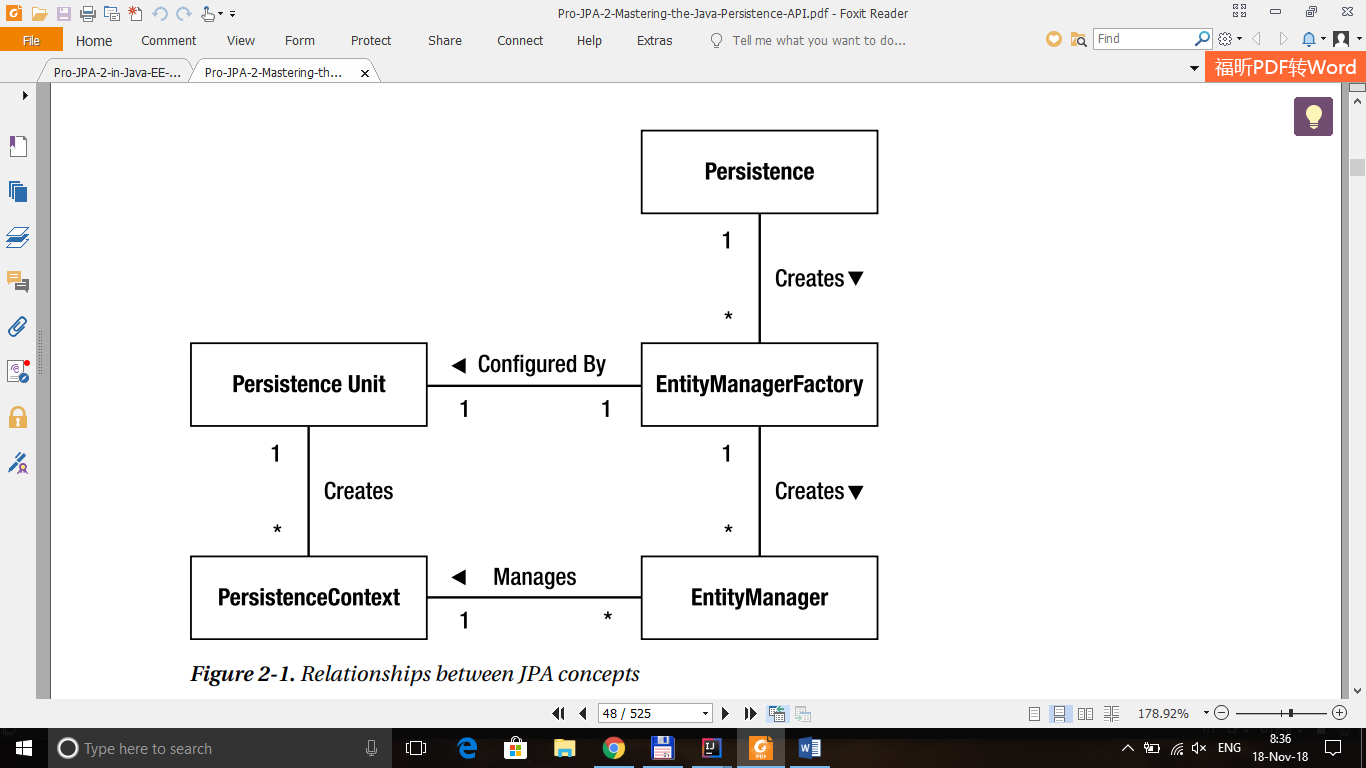
3. Modify \standalone\configuration\standalone.xml

<subsystem xmlns="urn:jboss:domain:datasources:5.0">  
<datasources>   
 <datasource jndi-name="java:jboss/datasources/MySQLDS" pool-name="MySQLDS" enabled="true" use-java-context="true" statistics-enabled="true">  
 <connection-url>jdbc:mysql://localhost:3306/data</connection-url>  
 <driver>mysql</driver>  
 <pool>  
 <min-pool-size>10</min-pool-size>  
 <max-pool-size>20</max-pool-size>  
 <prefill>true</prefill>  
 </pool>  
 <security>  
 <user-name>app</user-name>  
 <password>app</password>  
 </security>  
 </datasource>  
 <drivers>   
 <xa-datasource-class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-class>  
   
 </drivers>  
</datasources>  
</subsystem>

**JPA**

- JTA (Java Transaction API) is used in J2EE-managed applications in which the container is responsible for transaction propagation. For application-managed transactions, you can use RESOURCE\_LOCAL.

- All entity managers come from factories of type EntityManagerFactory.



**A. JPA Java SE Enviroment**

**1. Create a Entity :**

**Step 1 :**

EntityManagerFactory emf = Persistence.createEntityManagerFactory("primary");

**primary :** persistence unit (from <persistence-unit name=”primary”> in persistence.xml)

**Step 2 :**

EntityManager em = emf.createEntityManager();

**Step 3 :**

DiskDrive diskDrive = new DiskDrive();  
diskDrive.setVendor("F5");  
diskDrive.setValue(200);

**Step 4 :**

em.persist(diskDrive);  
em.getTransaction().commit();  
em.close();  
emf.close();

Calling persist() is all that is required to initiate it being persisted in the database.

Java SE environment need to begin and to commit the transaction

**2. Find Entity**

DiskDrive diskDriver = em.find(DiskDrive.class, id);

id : id of Object

find() method will returns null if wrong Object or id does not exist.

**3. Remove Entity**

DiskDrive diskDrive = em.find(DiskDrive.class, id);  
em.remove(diskDrive);

**4. Update Entity**

DiskDrive diskDrive = em.find(DiskDrive.class, id);  
if (diskDrive!= null) {  
 diskDrive.setValue(diskDrive.getValue() + 100);  
}

**B. JPA Java EE Enviroment**

## **What Is a Session Bean?** *(https://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html)*

A **session bean** encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. To access an application that is deployed on the server, the client invokes the session bean’s methods. The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.

A session bean is not persistent. (That is, its data is not saved to a database.)

There are three types of session bean: ***stateless*, *stateful*, and *singleton***.

A stateless session bean sets out to complete an operation within the lifetime of a single method.

The server manages the lifecycle of a stateless session bean.

The server will invoke the **PostConstruct** callback as soon as it has completed initializing all the container services for the bean.

The server invokes the **PreDestroy** callback immediately before the server releases the bean instance to be garbagecollected.

**Remote Business Interfaces**

Use a session bean with a local interface from a remote client

@Remote  
public interface DiskRemote {  
 public String value();  
}

**1. Create a Entity**

**Step 1 :**

Use@PersistenceContext