Top 6 Java 8 features you can start using now

Posted on November 8, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

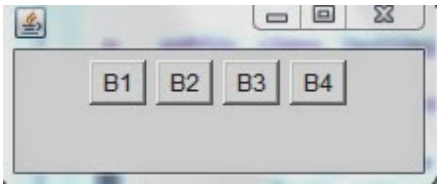
Unlike Java 7, Java 8 has some significant changes. You can get familiarised with the the following simple working code. All you need is Java 8 installed on your machine.

#1: Interface can have static and default methods. This tries to solve the diamond (aka multiple inheriance) issue. In the past it was essentially impossible for Java libraries to add methods to interfaces. Adding a method to an interface would mean breaking all existing code that implements the interface. Now, as long as a sensible default implementation of a method can be provided, library maintainers can add methods to these interfaces. The static

methods serve as utility methods, hence you don't need a separate helper class.

#2: Lambda expression for supporting closures, which simplifies your code. In the past, you need to write anonymous classes. Lambda expressions are **anonymous methods** which are intended to replace the bulkiness of **anonymous inner classes** with a much more compact mechanism.

Here is a basic example that covers **#1** and **#2**. A simple button click example.



Let's define an interface with a static method (e.g. a utility function) and two default methods as shown below.

```
1  package com.java8.examples;
2
3  import java.awt.event.ActionEvent;
4
5  public interface Test {
6
7   //Java 8: static method in an interface
8   static void doSomething(ActionEvent event) {
9          System.out.println("B1 clicked .....");
10     }
11
12  default void doAnotherSomething(ActionEvent eve
13         System.out.println("B2 clicked .....");
14     }
15
16
17  //Java 8: default method in interface
18  default void doWork(){
19   System.out.println("executing default method")
20  }
21 }
22
```

Define the *TestImpl* class, which implements the interface *Test*. The *TestImpl* uses closures via lambda expressions and makes use of the methods implemented in the interface Test.

```
package com.java8.examples;

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.awt.Button;
import javax.swing.JFrame;

public class TestImpl implements Test {

 public static void main(String[] args) {

  JFrame frame = new JFrame();
  frame.setLayout(new FlowLayout());

  Button button1 = new Button("B1");
  Button button2 = new Button("B2");
  Button button3 = new Button("B3");
  Button button4 = new Button("B4");

  frame.getContentPane().add(button1);
  frame.getContentPane().add(button2);
  frame.getContentPane().add(button3);
  frame.getContentPane().add(button4);

  //Java 8: Lambda expressions &amp;amp; invocation

  //invokes static method on the interface
  button1.addActionListener(Test::doSomething);
  Test testObj = new TestImpl();
  button2.addActionListener(testObj::doAnotherSo
  //input is e and result is println(..) and doW
  button3.addActionListener((e) -> {System.out.p
                                    testObj.doW

  //old way: pre Java 8: Anonymous inner class
  button4.addActionListener(new ActionListener()

   @Override
   public void actionPerformed(ActionEvent e) {
    System.out.println("B4 clicked");

   }
  });


  frame.setDefaultCloseOperation(JFrame.EXIT_ON_
  frame.pack();
  frame.setVisible(true);
 }
}
```

If you click on all 4 buttons from B1 to B4, you will get an output of

```
B1 clicked .....
B2 clicked .....
```

16 Technical Key Areas

open all | close all

- Best Practice (6)
- Coding (26)
- Concurrency (6)
- Design Concepts (7)
- Design Patterns (11)
- Exception Handling (3
- Java Debugging (21)
- Judging Experience I
- Low Latency (7)
- Memory Managemen
- Performance (13)
- QoS (8)
- Scalability (4)
- SDLC (6)
- Security (13)
- Transaction Managen

80+ step by step Java Tutorials

open all | close all

- Setting up Tutorial (6)
- Tutorial - Diagnosis (2
- Akka Tutorial (9)
- Core Java Tutorials (2
  - Debugging Tutorial
  - Java 8 Tutorial (4)
    - 02: ♦ Java 8 Stre
    - Learning to write
    - Top 6 Java 8 fea
    - Understanding J
  - JSON and Java Tu
  - XML and Java Tutc
  - CSV and Java Tutc
- Hadoop & Spark Tuto

```
3 B3 clicked ...
4 executing default method
5 B4 clicked
6
```

Q. How does " button3.addActionListener((e) -> {System.out.println("B3 clicked …"); testObj.doWork(); }); " know what method to execute? Did you notice that there is no new *ActionListener( )* and no *public void actionPerformed(ActionEvent event)*?

A. Here the compiler understands that there is only one method in the *ActionListener* interface and that takes only one parameter. So obviously, it decides (finalizes) that the e as a reference to *ActionEvent*.

The functional interfaces that are annotated with *@FunctionalInterface* in Java 8 ensures that you can only define one abstract method. You can define as many default and static method implementations as you want.

#3 Java 8 adopts Joda as its native date & time framework. Due to what was perceived as a design flaw in Joda, Java 8 implemented its own new date / time API from scratch. The new APIs were designed with simplicity in mind.

All the core classes in the new API are constructed by fluent factory methods. When constructing a value by its constituent fields, the factory is called of; when converting from another type, the factory is called from.

```
1  package com.java8.examples;
2
3  import java.time.LocalDate;
4  import java.time.LocalDateTime;
5  import java.time.LocalTime;
6  import java.time.Month;
7
8  public class TimeTest {
9
10  public static void main(String[] args) {
11   // The current date and Time
12   LocalDateTime timePoint = LocalDateTime.now();
13   System.out.println("now=" + timePoint);
14
15   //10th of Jan  2012
16   LocalDate pastDate = LocalDate.of(2012, Month.
17   System.out.println("pastDate=" + pastDate);
```

JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

```
18
19
20    LocalTime mathsClassTonight = LocalTime.of(19,
21    System.out.println("mathsClassTonight=" +  mat
22
23    mathsClassTonight = LocalTime.parse("19:30:50"
24    System.out.println("mathsClassTonight=" +  mat
25
26
27    //add 12 days to pastDate
28    LocalDate plusDays = pastDate.plusDays(12);
29    System.out.println("pastPlus12days=" +  plusDa
30
31
32    LocalDate epochDay = LocalDate.ofEpochDay(150)
33    System.out.println("epochDay=" +  epochDay);
34
35   }
36
37 }
38
```

The **output** will be:

```
1 now=2014-04-02T16:27:41.467
2 pastDate=2012-01-10
3 mathsClassTonight=19:30
4 mathsClassTonight=19:30:50
5 pastPlus12days=2012-01-22
6 epochDay=1970-05-31
7
```

Example 1: The ***java.lang.Runnable*** interface has been made a functional interface.

```
1  package com.java8.examples;
2
3  public class Example1 {
4
5   public static void main(String[] args) {
6     System.out.println("main thread: " + Thread.c
7
8     //run method takes no arguments so: () -> (bo
9     //closure worker thread1
10    Runnable r = () -> (System.out.println("worke
11    new Thread(r).start(); //pass the Runnable cl
12
13    //closure worker thread2
14    new Thread( () -> (System.out.println("worker
15   }
16 }
17
18
```

Output:

main thread: main
worker thread: Thread-0
worker thread: Thread-1

Example 2: The **java.util.function package** has a number of functional interfaces.

```
1  package com.java8.examples;
2
3  import java.util.function.Consumer;
4
5  public class Example2 {
6
7    public static void main(String[] args) {
8     //closure of Consumer 1. the accept(T t) meth
9     Consumer<Integer>  c1 = (x) -> {System.out.pr
10    //closure of Consumer 2. x is an Integer
11    Consumer<Integer>  c2 = (x) -> {System.out.pr
12
13
14    c1.accept(5);
15    c2.accept(5);
16
17    //andThen is a default method implementation
18    Consumer<Integer> c3 = c1.andThen(c2);
19    c3.accept(6);
20   }
21 }
22
23
```

Output:

consumed: 5
consumed: 25
consumed: 6
consumed: 36

Example 3: The ***Function*** is similar to ***Consumer***, but the *Function* interface has a return value in its *apply(T t)* method, whereas the *Consumer* has a void method *accept(T t)*.

```
1  package com.java8.examples;
2
3  import java.util.function.Function;
4
```

```
5  public class Example3 {
6
7    public static void main(String[] args) {
8     //closure of Function.  apply(T t) method tak
9     Function<Double, Double>  c1 = (arg) -> (arg
10    Function<Double, Double>  c2 = (arg) -> (arg
11
12    Double result1 = c1.apply(5.0);
13    Double result2 = c2.apply(5.0);
14
15    System.out.println("result1= " + result1);
16    System.out.println("result2= " + result2);
17
18    //andThen is a default method implementation
19    Function<Double, Double> c3 = c1.andThen(c2);
20    Double result3 = c3.apply(6.0);
21
22    System.out.println("result3= " + result3);//
23
24   }
25 }
26
```

Output:

result1= 10.0
result2= 25.0
result3= 144.0

Example 4: The ***Predicate*** interface from *java.util.function* package has *test(T t)* method that returns a boolean value indicating test condition has passed or failed. The predicates are often used to filter a collection of objects.

```
1  package com.java8.examples;
2
3  import java.util.Arrays;
4  import java.util.List;
5  import java.util.function.Predicate;
6
7  public class Example4 {
8
9   private static final int SHORT_NAME_LENGTH = 3;
10
11  public static void main(String[] args) {
12
13   List<String> languages = Arrays.asList("Java",
14
15   //The Predicate interface has test(T t) that r
16   Predicate<String> shortNamePredicate = (s) ->
17
18
19   Predicate<String> startsWithCPredicate = (s) -
20
21   languages.stream()
22            .filter(shortNamePredicate)
23            .forEach(e -> System.out.println("sho
```

```
24
25    languages.stream()
26              .filter(startsWithCPredicate)
27              .forEach(e -> System.out.println("sta
28
29   }
30 }
31
```

Output:

short names: C++
short names: PHP
short names: C#
starts with J: Java
starts with J: JRuby
starts with J: Jython

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 04: ♥♦ Top 6 tips to transforming your thinking from OOP to FP with examples

Java 8: Different ways to sort a collection of objects in pre and post Java 8 ›

Posted in Java 8, Java 8 Tutorial

Tags: TopX

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

↑