УДК 00.004.4

Y.E. Parfyonov, V.M. Fedorchenko

Simon Kuznets Kharkiv National University of Economics, Kharkiv

USING NEW FEATURES OF JAVA SE 8

The paper analyzes new features of Java SE 8, and their using for application development. Discussed date and time representing, lambda-expressions, references to methods, default methods. Considered basic features of the Stream API and their application to work with collections in Java SE 8. Noted new possibilities of JavaFX 8 technology to design, build, test, debug and deploy rich client applications that work consistently for different target platforms.

Keywords: *platform, JIT compiler, Java Virtual Machine, Java Development Kit, lambda expressions, method references, default methods, collections, JavaFX 8, Date-Time API, FXML, XML, CSS, 3D graphics, multi-touch, immutable-value classes, domain-driven design.*

Introduction

As known Java platform is a set of several computer software and specifications initially developed by Sun Microsystems, which provides tools for developing application software and deploying it in a cross-platform computing environment.

Java is widely used in a lot of computing platforms from embedded devices and mobile phones to enterprise servers and supercomputers [1].

The heart of the Java platform is the concept of a "virtual machine" that executes Java bytecode programs. This bytecode is the same no matter what hardware or operating system the program is running under.

There is a JIT compiler within the Java Virtual Machine, or JVM. The JIT compiler translates the Java bytecode into native processor instructions at run-time and caches the native code in memory during execution.

The use of bytecode as an intermediate language permits Java programs to run on any platform that has a virtual machine available.

There are several editions of java platform, each targeting a different class of devices:

- *Java Platform, Micro Edition* (Java ME): specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. Often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.
- *Java Platform, Standard Edition* (Java SE): for general-purpose use on desktop PCs, servers and similar devices.
- *Java Platform, Enterprise Edition EE* (Java EE): Java SE plus various APIs useful for multi-tier client-server enterprise applications.

The main part

Java SE is a platform for development and deployment of portable applications for desktop and server environments. It defines a wide range of general purpose APIs – such as Java APIs for the Java Class Library – and includes the Java Language Specification and the Java Virtual Machine Specification. One of the most well-known implementations of Java SE is Oracle Corporation's Java Development Kit (JDK) [2].

From the very beginning, Java SE used the object-oriented Java programming language as its single target language. Actually, it was listed as a core part of the Java platform not so much time ago. Therefore the language and runtime were commonly considered a single unit.

However, many modern programming languages support several programming paradigms and are typically outside of the scope of the phrase "platform". It allows improving the "platform" and the languages independently. So, an effort was made with the Java 7 specification to more clearly treat the Java language and the Java virtual machine as separate entities, so that they are no longer considered a single unit.

Java has undergone several changes since the release of JDK 1.0 on, as well as numerous additions of classes and packages to the standard library.

On February 19, 1997 Sun Microsystems released JDK 1.1. Major additions included an extensive retooling of the AWT event model, inner classes added to the language, JavaBeans and JDBC.

J2SE 1.2 was released on December 8, 1998. This and subsequent releases through J2SE 5.0 were rebranded Java 2 and the version name "J2SE" (Java 2 Platform, Standard Edition) replaced JDK. Major additions included reflection, collections framework, Java

IDL (an interface description language implementation for CORBA interoperability), and the inte-

gration of the Swing graphical API into the core classes. A Java Plug-in was released, and Sun's JVM was equipped with a JIT compiler for the first time.

Notable changes in J2SE 1.3 (May 8, 2000) included the bundling of the HotSpot JVM, JavaSound, Java Naming and Directory Interface and Java Platform Debugger Architecture.

J2SE 1.4 (February 6, 2002) became the first release of the Java platform developed under the Java Community Process as JSR 59. Major changes included regular expressions modeled after Perl, exception chaining, an integrated XML parser and XSLT processor (JAXP), and Java Web Start.

J2SE 5.0 (September 30, 2004) added several significant new language features including the for-each loop, generics, autoboxing and var-args.

Java SE 6 (December 11, 2006) bundled with a database manager and facilitates the use of scripting languages with the JVM (such as JavaScript). As of this version, Sun Microsystems replaced the name "J2SE" with Java SE and dropped the ".0" from the version number. Other major changes include support for pluggable annotations, many GUI improvements, including native UI enhancements to support the look and feel of Windows Vista, and improvements to the Java Platform Debugger Architecture & JVM Tool Interface for better monitoring and troubleshooting.

Java SE 7 (July 28, 2011) added many small language changes including strings in switch, try-with-resources and type inference for generic instance creation.

The JVM was extended with support for dynamic languages like Clojure, Groovy, Scala and so on, while the class library was extended among others with a join/fork framework, an improved new file I/O library and support for new network protocols such as SCTP.

Java SE 8 is a major feature release of Java programming language development. Its initial version was released on March 18, 2014.

In the release entire new APIs, have evolved, and many of the original JDK 1.0 classes and methods have been deprecated.

Moreover, the changes to Java 8 are in many ways more profound than any other changes to Java in its history.

They enable developers to write programs more easily - instead of writing verbose code [3].

Top Features and enhancements in Java SE 8 relate to [4]:

- Java Programming Language:
- Lambda Expressions;
- Method references;
- Default methods;
- Collections API;
- JavaFX technology;
- Date-Time API.

Lambda expressions

There is a well-known issue in software engineering connected with frequent changes of user requirements.

To address the issue different approaches can be applied. A widely spread technique is making a block of code available without executing it. This block of code can be called later by other parts of a program. Passing code is a way to give new behaviors as arguments to a method.

Prior to Java 8 anonymous classes were a mean to get rid of the some verbosity associated with declaring multiple concrete classes for an interface that are needed only once.

Java 8 introduces new feature that resolves this problem: lambda expressions, which let you represent a behavior or pass code in a concise way [5].

A lambda expression is an unnamed function with parameters and a body [6]:

() -> "hi" takes no parameters and returns a string

(int x) -> x + 1 takes an int parameter and returns the parameter value incremented by 1

(int x, int y) -> x + y takes two int parameters and returns the sum

(String msg) -> {System.out.println(msg);} takes a String parameter and prints it on the standard output.

In general, the lambda expressions allow developers to pass logic in a compact way.

For example, consider the following code:

```
button.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     System.out.println("Hello!");
   }});
```

It uses an anonymous inner class to add event handler for a button click action. The action handler prints out a message when that the button is clicked.

By using a lambda expression, we can add the action handler to a button click event in a single line of code:

```
button.addActionListener(e->
System.out.println("Hello!"));
```

Thus, lambda expressions support treating functionality as a method argument, or code as data. They let Java developers express instances of single-method interfaces (referred to as functional interfaces) more compactly.

Method references

Method references are an important feature related to lambda expressions. They provide access to a method without executing it.

Lambda expressions ensure similar but more powerful capabilities.

Sometimes, however, a lambda expression just calls an existing method. In those cases, it is often

clearer to refer to the existing method by name using method references.

There are two main types of method references:

Reference to a static method (ClassName::StaticMethodName)

Reference to an instance method of an object (ObjectReferenceName::InstanceMethodName)

Instance Method References Of Classes

Therefore, method references are compact, easy-to-read lambda expressions for methods that already have a name.

Default methods

Default methods allow developers to add new methods to interfaces without breaking their existing implementation. An interface can define implementation, which will be used as default in the situation where a concrete class fails to provide an implementation for that method.

Therefore, default methods have introduced as a mechanism to extending interfaces in a backward compatible way.

Collections API

Collections are fundamental to many programming tasks because they group some variable number of data items that have some shared significance in a domain and as a rule need to be operated upon together.

However processing collections was far from perfect in Java: it was necessary to use a loop or obtain an iterator for a collection and process elements of the collections in a sequence. For example, to print only the elements meeting some condition after they would be altered in some way, we had to write something like that:

```
List<String> myList = Arrays.asList("a1", "a2", "b1", "c2", "c1");
List<String> temp = new ArrayList<String>();
for (String element : myList)
  if (element.startsWith("c"))
    temp.add(element.toUpperCase());
Collections.sort(temp);
for (String element : temp)
  System.out.println(element);
```

Fortunately, in the Java SE 8 into the Collections API the new Stream API is integrated (java.util.stream package). It enables bulk operations on collections, such as sequential or parallel map-reduce transformations. So, previous code can be more clear and simple:

```
List<String> myList = Arrays.asList("a1", "a2", "b1", "c2", "c1");
myList.stream()
.filter(s -> s.startsWith("c"))
.map(String::toUpperCase)
.sorted()
.forEach(System.out::println);
```

JavaFX technology

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms such as embedded devices, smartphones, TVs, tablet computers, and desktops. Typically, the applications display information in a high-performance modern user interface that features audio, video, graphics, and animation [7].

JavaFX 8 release significantly increases power of JavaFX technology. It incorporates many useful features: improved XML-based declarative markup language FXML for constructing user interface; all the major UI controls, which can be skinned with standard Web technologies such as CSS; 3D graphics, multitouch and hardware-accelerated graphics support.

JavaFX 8 is included in JDK 8 and is the officially recommended graphics toolkit for Java 8 applications.

Date Time API

Since Java SE was first released, the main support for dates and times in Java was the java.util.Date class. However, actually it does not represent a “date” in a human-friendly way. It does represent only an instantaneous point in time based on millisecond precision, measured from the January 1, 1970 [8].

This issue is not something the average developer would expect to deal with when writing date-handling code. In addition, some of the date and time classes also exhibit quite poor API design.

In order to address these problems a new date and time API has been designed for Java SE 8 [9].

The new API is driven by three core ideas:

- **Immutable-value classes.** One of the serious weaknesses of the existing formatters in Java is that they are not thread-safe. The new API avoids this issue by ensuring that all its core classes are immutable and represent well-defined values.

- **Domain-driven design.** The new API models its domain very precisely with classes that represent different use cases for Date and Time closely. For example, using date and time independently, representing the one in “human” terms or as an instant on the timeline etc.

- **Separation of chronologies.** The new API allows people to work with different calendaring systems in order to support the needs of users in some areas of the world, such as Japan or Thailand, that don’t necessarily follow ISO-8601.

The new java.time API consists of five packages [8]:

- java.time – the base package containing the value objects;

- java.time.chrono – provides access to different calendar systems;

- java.time.format – allows date and time to be formatted and parsed;

- java.time.temporal – the low level framework and extended features;

- java.time.zone – support classes for time-zones.

There about 70 new public types in the packages. The most important ones are the base and format packages. The java.time package provides such classes as LocalDate (immutable value type that represents a date without time-of-day or time-zone), LocalTime (a value type with no associated date or time-zone), LocalDateTime (represents a date with time-of-day part), Instant (a "machine" view of time as a point on the time-line without any other contextual information). Moreover, it includes many other useful public types.

The java.time.format package is related to formatting and printing dates and times.

One of its valuable elements is DateTimeFormatter class. It provides three kinds of formatters to print a date/time value: predefined standard formatters, locale-specific formatters and formatters with custom patterns.

Thereby the new Date Time API brings adequate support for the date and time, which largely simplifies developing the Java applications.

References

1. Java (software_platform). [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Java (software_platform).

2. Java Platform, Standard Edition. [Electronic resource]. – Access mode: en.wikipedia.org/wiki/Java_Platform,_Standard_Edition.

3. Java SE 8. [Electronic resource]. – Access mode: http://www.tutorialspoint.com/java8/java8_overview.htm.

4. What's New in JDK 8. [Electronic resource]. – Access mode: http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html.

5. Raoul-Gabriel Urma. Java 8 in Action: Lambdas, streams, and functional-style programming / Raoul-Gabriel Urma, Mario Fusco, Alan Mycroft. – Manning Publications, 2014. – 497 p.

6. Liguori R. Java 8 Pocket Guide / R. Liguori, P. Liguori. – O'Reilly Media, Inc., 2014. – 223 p.

7. JavaFX: Getting Started with JavaFX. [Electronic resource]. – Access mode: http://docs.oracle.com/ javase/8/javafx/get-started-tutorial/jfx-overview.htm.

8. Intuitive, Robust Date and Time Handling, Finally Comes to Java. [Electronic resource]. – Access mode: http://www.infoq.com/articles/java.time.

9. Java SE 8 Date and Time. [Electronic resource]. – Access mode: http://www.oracle.com/technetwork/ articles/java/jf14-date-time-2125367.html.

Надійшла до редколегії 13.02.2015

Рецензент: канд. екон. наук, проф. І.О. Золотарьова, Харківський національний економічний університет ім. С. Кузнеця, Харків.

ВИКОРИСТАННЯ НОВИХ МОЖЛИВОСТЕЙ JAVA SE 8

Ю.Е. Парфьонов, В.М. Федорченко

У роботі аналізуються нові можливості платформи Java SE 8 та їх застосування для розробки застосувань. Розглянуто подання дати та часу, використання лямбда-виразів, посилань на методи, методів за замовчуванням. Викладені основні особливості Stream API щодо роботи з колекціями в Java SE 8. Відзначено нові можливості технології JavaFX 8, для проектування, створення, тестування, налагодження та розгортання насичених клієнтських застосувань, які злагоджено працюють на різних цільових платформах.

Ключові слова: *платформа, JIT compiler, Java Virtual Machine, Java Development Kit, лямбда-вирази, посилання на методи, методи за замовчуванням, колекції, JavaFX, Date-Time API, FXML, XML, CSS, 3D-графіка, multi-touch, immutable-value класи, проблемно-орієнтоване проектування.*

ИСПОЛЬЗОВАНИЕ НОВЫХ ВОЗМОЖНОСТЕЙ JAVA SE 8

Ю.Э. Парфенов, В.Н. Федорченко

В работе анализируются новые возможности платформы Java SE 8 и их применение для разработки приложений. Рассмотрены представление даты и времени, использование лямбда-выражений, ссылок на методы, методов по умолчанию. Изложены основные особенности Stream API для работы с коллекциями в Java SE 8. Отмечены новые возможности технологии JavaFX 8 для проектирования, создания, тестирования, отладки и развертывания насыщенных клиентских приложений, которые слаженно работают на различных целевых платформах.

Ключевые слова: *платформа, JIT compiler, Java Virtual Machine, Java Development Kit, лямбда-выражения, ссылки на методы, методы по умолчанию, коллекции, JavaFX 8, Date-Time API, FXML, XML, CSS, 3D-графика, multi-touch, immutable-value классы, проблемно-ориентированное проектирование.*