DSA Assignment #1 (IS2)

7 messages

Allyson Mae Tubtub <allysonmae17@gmail.com> Fri, Jul 1, 2016 at 10:34 AM
To: arrowbrave@gmail.com

Tubtub, Allyson Mae J.
BSIS - 2 (irreg)
Assignment in DSA

1. History of Java

Java, having been developed in 1991, is a relatively new programming language. At that time, James Gosling from Sun Microsystems and his team began designing the first version of Java aimed at programming home appliances which are controlled by a wide variety of computer processors.

Gosling's new language needed to be accessible by a variety of computer processors. In 1994, he realized that such a language would be ideal for use with web browsers and Java's connection to the internet began. In 1995, Netscape Incorporated released its latest version of the Netscape browser which was capable of running Java programs.
Why is it called Java? It is customary for the creator of a programming language to name the language anything he/she chooses. The original name of this language was Oak, until it was discovered that a programming language already existed that was named Oak. As the story goes, after many hours of trying to come up with a new name, the development team went out for coffee and the name Java was born.

2. How is Java platform independent?
Java code is platform independent in the sense that the same Java application or algorithms (typically compiled to Java bytecode and packaged in a .jar file) will run identically on Windows and Linux.
A Java code will run on any JVM (Java Virtual Machine). Literally you can run same Java code on Windows JVM, Linux JVM, Mac JVM or any other JVM practically and get same result every time.
Java libraries (e.g. all the nice open source toolsets) are usually platform independent, as long as they are written in pure Java. Most libraries try to stick with pure Java in order to maintain platform independence, but there are some cases where this is not possible (e.g. if the library needs to interface directly with special hardware, or call a C/C++ library that uses native code).
The Java platform / runtime environment is platform independent in the sense that the same libraries (images, networking, File IO etc.) are available and work in the same way on all platforms. This is done deliberately in order to allow applications that use these libraries to be able to run on any platform. For example, the Java libraries that access the filesystem know the fact that Windows and Linux use different filename path separators, and take account of this for you. Of course, this means that under the hood the runtime environment does make use of platform-specific features, so you need a different JRE for each platform.
The JVM itself (i.e. the Java Virtual Machine that is responsible for JIT compililng and running Java bytecode) is platform independent in the sense that it is available on many platforms (everything from mainframes to mobile phones). However specific versions of the JVM are needed for each underlying platform to take account of different native instruction codes and machine capabilities (so you can't run a Linux JVM on Windows and vice Versa). The JVM is packaged as part of the Java platform / runtime environment as above.
Overall, Java is probably about as close to true platform independence as you can get, but as you can see there is still quite a bit of platform-specific work done under the hood.

3. Differentiate JDK, JRE, and JVM.
JVM is Java Virtual Machine -- the JVM actually runs Java bytecode.
As we all aware when we compile a Java file, output is not an exe but its a .class file. .class file consists of Java byte codes which are understandable by JVM. Java Virtual Machine interprets the byte code into the machine code depending upon the underlying operating system and hardware combination. It is responsible for all the things like garbage collection, array bounds checking, etc JVM is platform dependent.
The JVM is called virtual because it provides a machine interface that does not depend on the underlying operating system and machine hardware architecture. This independence from hardware and operating system is a cornerstone of the write-once run-anywhere value of Java programs.
There are different JVM implementations are there. These may differ in things like performance, reliability, speed, etc. These implementations will differ in those areas where Java specification doesnt mention how to implement the features, like how the garbage collection process works is JVM dependent, Java spec doesnt define any specific way to do this.

JDK is Java Developer Kit -- the JDK is what you need to compile Java source code.
Java Developer Kit contains tools needed to develop the Java programs, and JRE to run the programs. The tools include compiler (javac.exe), Java application launcher (java.exe), Appletviewer, etc
Compiler converts java code into byte code. Java application launcher opens a JRE, loads the class, and invokes its main method.
You need JDK, if at all you want to write your own programs, and to compile them. For running java programs, JRE is sufficient.
JRE is targeted for execution of Java files. i.e. JRE = JVM + Java Packages Classes(like util, math, lang, awt,swing etc)+runtime libraries.
JDK is mainly targeted for java development. I.e. You can create a Java file (with the help of Java packages), compile a Java file and run a java file.

JRE is Java Runtime Environment -- is what you need to run a Java program and contains a JVM, among other things.
Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc. Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE. If you want to run any java program, you need to have JRE installed in the system
The Java Virtual Machine provides a platform-independent way of executing code; That mean compile once in any machine and run it any where(any machine).

4. Why was the main method declared static?
First of all, if we write static before any method, it basically means that , this method has only one instance, regardless of how many objects we create for that particular class. So, if we have a class X and we have a static method cool(), then if we create different objects of class X , we shall still access the same cool() method from all the different objects. So, it makes more sense to write X.cool() to call a static method of the class. Since we are calling the same instance of the function from all the different objects.
Now, the main method lives inside the class which is also the name of our java file and main() is basically the entry point of our whole project consisting of multiple classes and files. But in java everything is a class, so in order to call a method, we have to create an object of that class, but main should be called before building any object or any instance of the class since it's the entry point. That is why we name the file same as the class and make main static so that JVM can call main without creating the instance of the class , but knowing the name of the class. And I told you that the static method can be called by only knowing the name of the class, no need to create an object.

5. Is Java purely object-oriented? Explain.
I think Java is object-oriented but not purely because this programming language has some of the characteristics of being an object-oriented program.
Java is object-oriented which means software developed in Java are combination of different types of object, and those Objects have ways of interacting with each other.
Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:
-Object
-Class
-Inheritance
-Polymorphism
-Abstraction
-Encapsulation

Object-oriented programming (OOP) languages are designed to overcome these problems.
-The basic unit of OOP is a class, which encapsulates both the static properties and dynamic operations within a "box", and specifies the public interface for using these boxes. Since classes are well-encapsulated, it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.
-OOP languages permit higher level of abstraction for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++ and C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.

Allyson Mae Tubtub <allysonmae17@gmail.com> Fri, Jul 1, 2016 at 10:37 AM
To: arrowbrave@gmail.com

Tubtub, Allyson Mae - BSIS2 DSA.doc
31K

arrowbrave <arrowbrave@gmail.com> Fri, Jul 1, 2016 at 11:49 AM
To: allysonmae17@gmail.com

received

Sent from my Mi phone
[Quoted text hidden]

---

arrowbrave <arrowbrave@gmail.com> Fri, Jul 1, 2016 at 11:49 AM
To: allysonmae17@gmail.com

received

Sent from my Mi phone
On Allyson Mae Tubtub <allysonmae17@gmail.com>, Jul 1, 2016 10:37 wrote:

Allyson Mae Tubtub <allysonmae17@gmail.com> Tue, Jul 5, 2016 at 8:12 AM
To: arrowbrave <arrowbrave@gmail.com>

WAD No.1
Tubtub, Allyson Mae J. (BSIS-II irreg)

Tubtub, Allyson Mae_BSIS_2.zip

[Quoted text hidden]

R. O. Valente <arrowbrave@gmail.com> Tue, Jul 5, 2016 at 8:12 AM
To: allysonmae17@gmail.com

Email received. Thanks. :)

Allyson Mae Tubtub <allysonmae17@gmail.com> Tue, Jul 5, 2016 at 8:18 AM
To: "R. O. Valente" <arrowbrave@gmail.com>

Salamat po sa Dios :)

On Tue, Jul 5, 2016 at 8:12 AM, R. O. Valente <arrowbrave@gmail.com> wrote:

> Email received. Thanks. :)