Introduction to Java

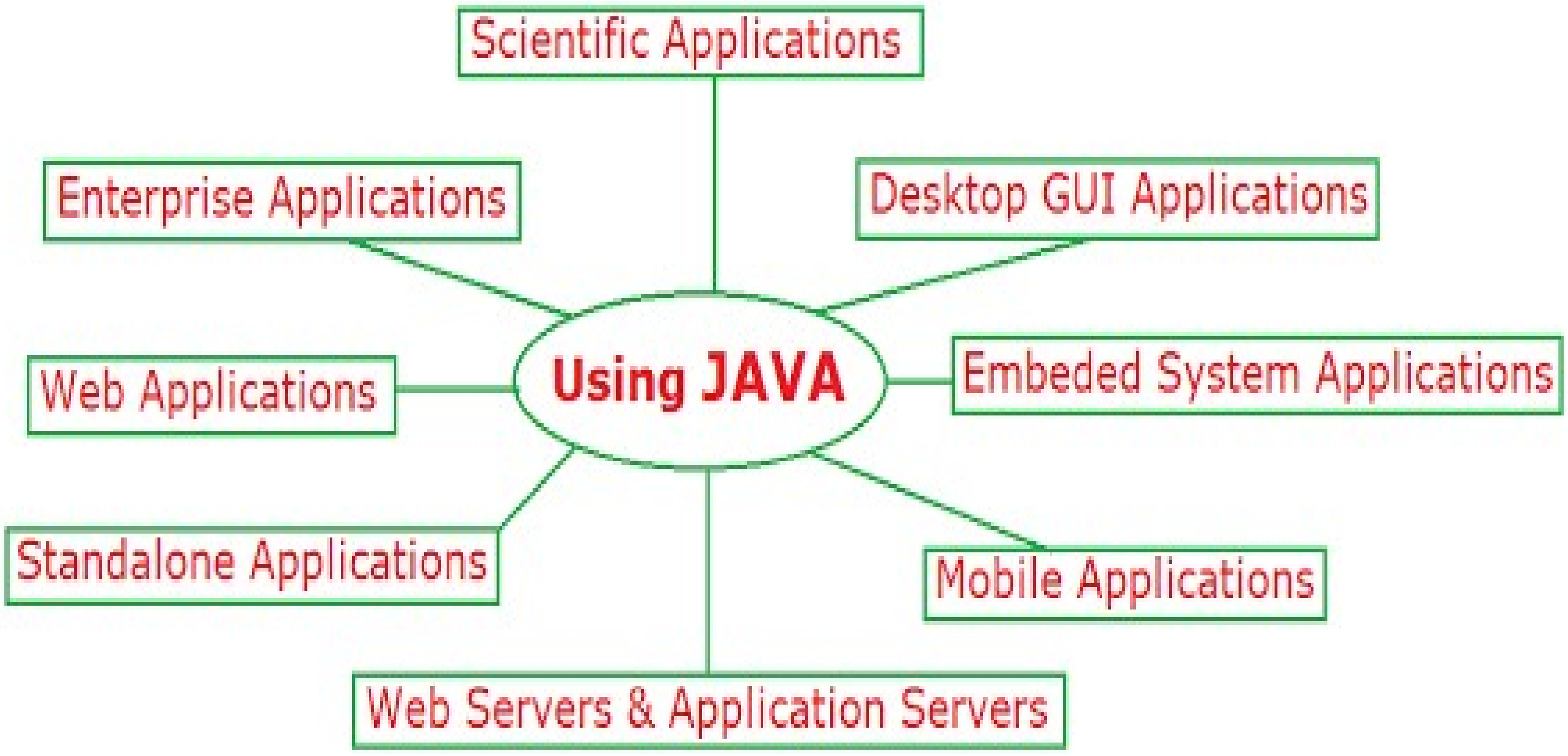
Learning Outcomes

- Understand basic terminologies and concepts of Java
  - Java programming Environment and Runtime Environment,
  -
  - Java Virtual Machine (JVM), Java compiler, Bytecode, Java Buzzwords, Java program structure, Comments, Lexical Issues

What is Java?

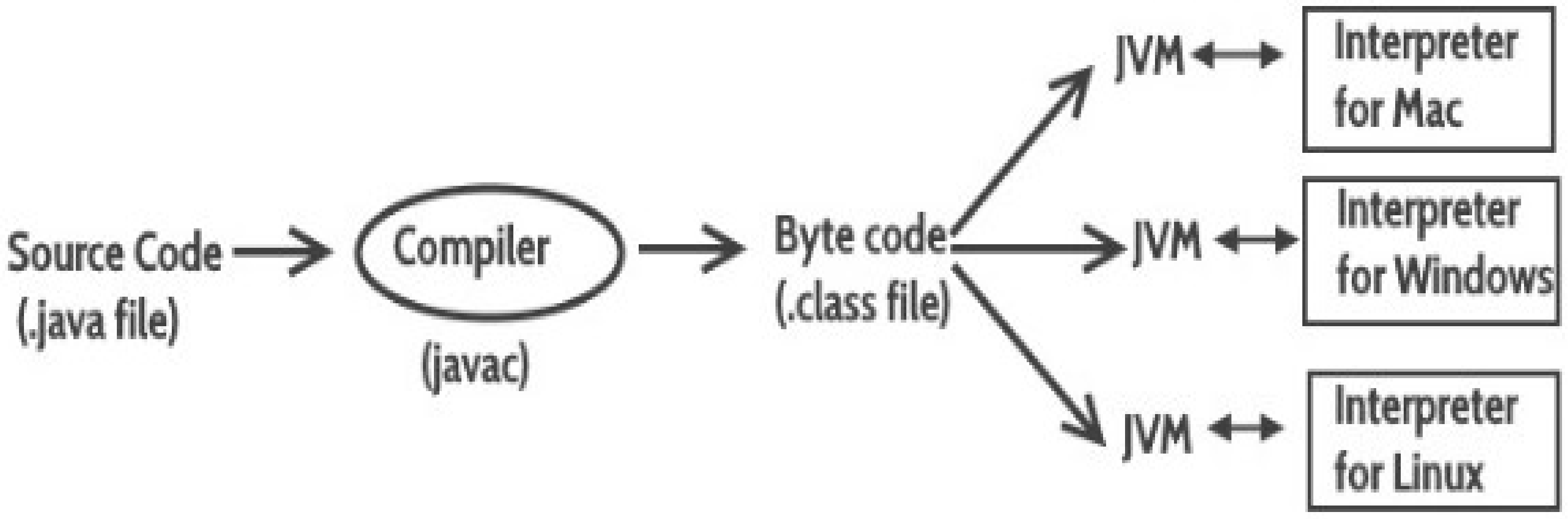
- Java is one of the most popular programming language in the world.
- It is initially called “Oak,” but was renamed “Java” in 1995.
- It is owned by Oracle, and more than 3 billion devices run Java.
- It works on different platforms (Windows, Mac, Linux)
- It is open-source and free.
- Java is guaranteed to be Write Once, Run Anywhere.

Java is used for?



What happens to Java Source Code

- The ***javac*** compiler takes java program (**.java** file containing source code) and translates it into machine code (referred as **byte code** or **.class** file).



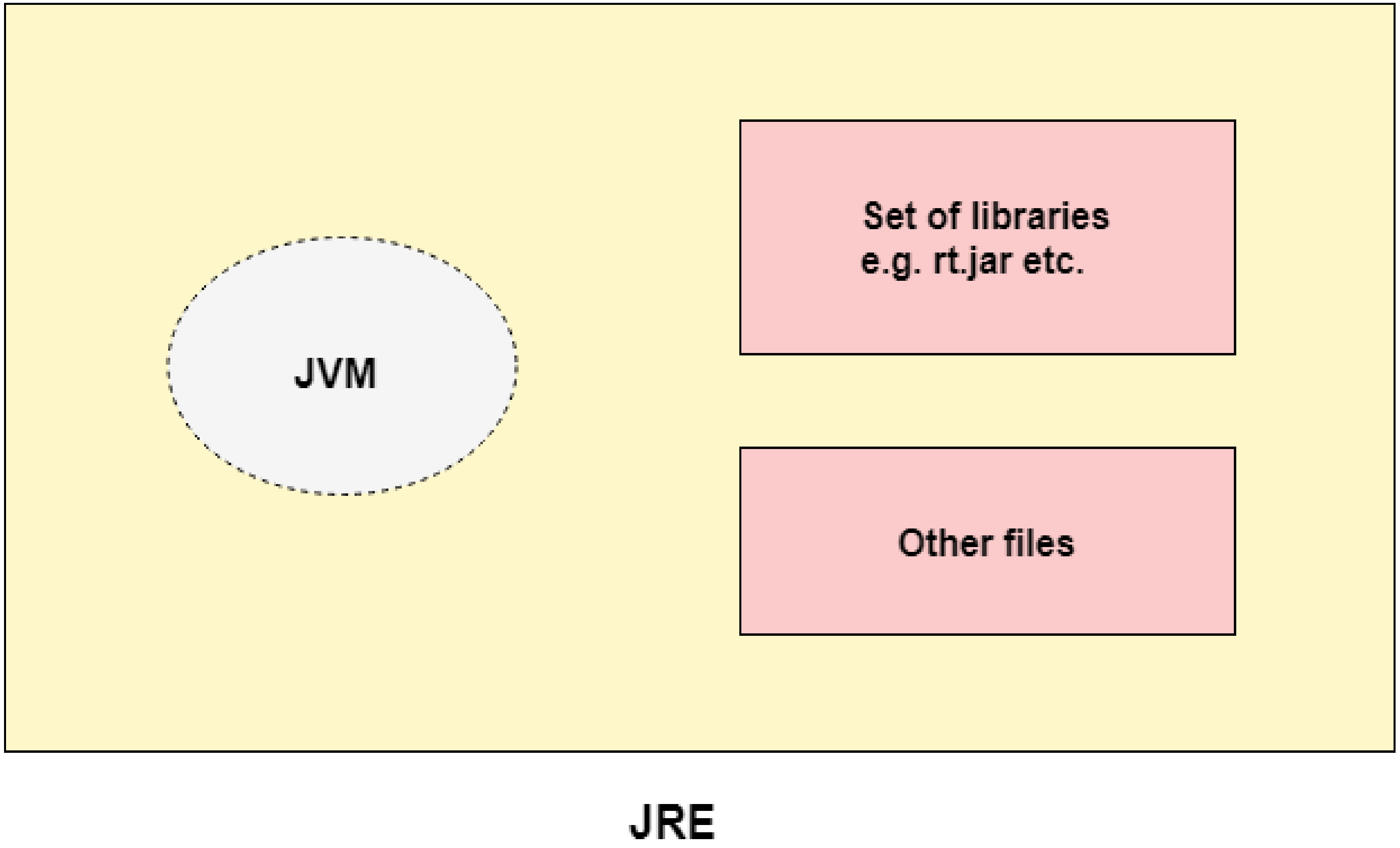
Java Virtual Machine(JVM)

- A specification that provides a runtime environment in which Java bytecode can be executed.
- It can run programs which are compiled to Java bytecode.
- JVMs are available for many hardware and software platforms.
- JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.
- The JVM performs the following main tasks:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

Java Runtime Environment (JRE)

- The JRE is the software environment in which programs compiled for a typical JVM implementation can run.
- It provides the runtime environment. It is the implementation of JVM. It physically exists. It contains a
- The runtime system includes:
  - A set of libraries + other files that JVM uses at runtime.
  - Implementation of the JVM

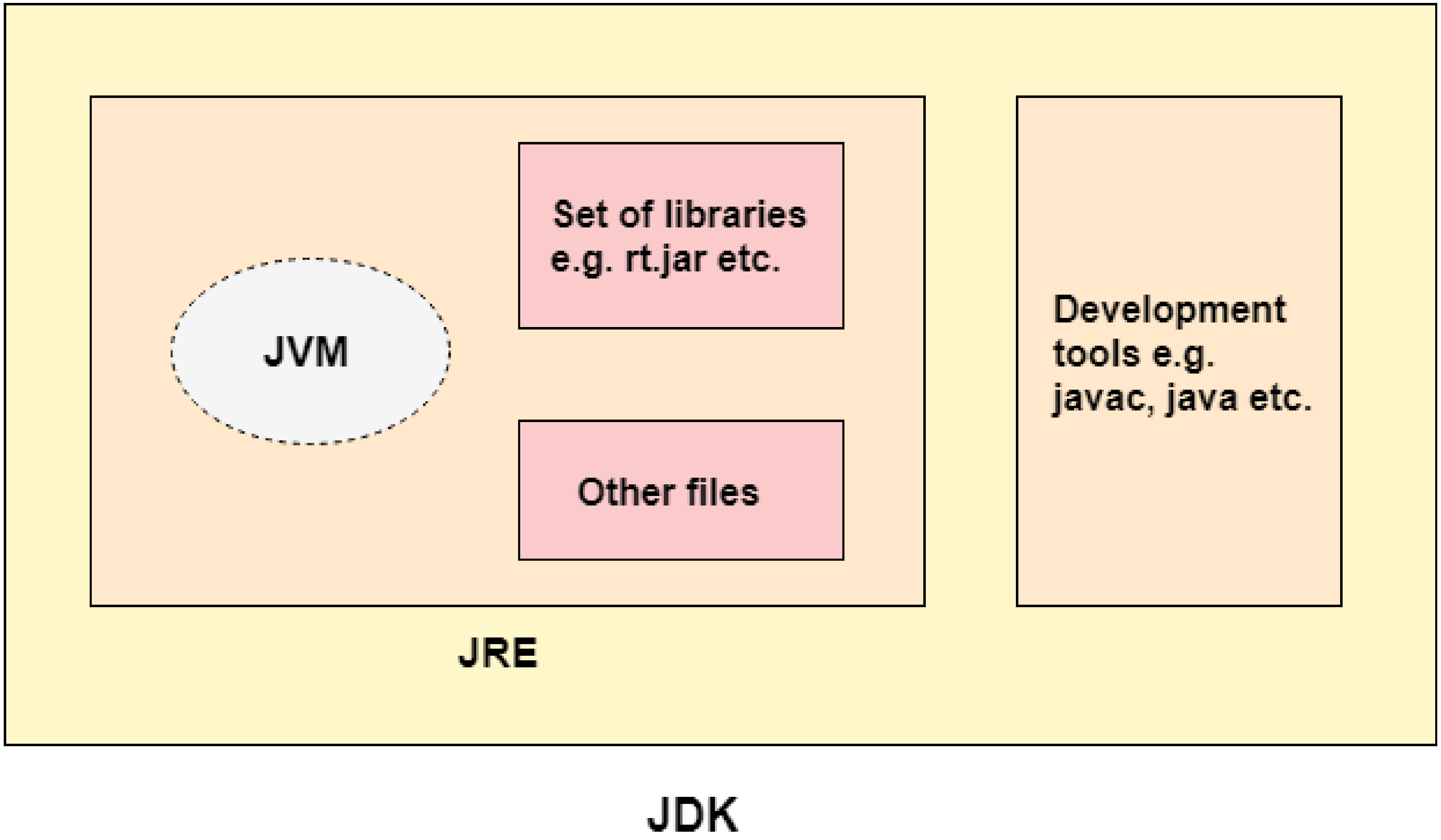
Java Runtime Environment (JRE)



Java Development Kit (JDK)

- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets
- It contains JRE + development tools.
- The JDK contains a Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

Java Development Kit (JDK)


Set of libraries e.g. rt.jar etc.
JVM
Other files
Development tools e.g. javac, java etc.
JRE
JDK

Java Bytecode

- Java bytecode is the instruction set for the Java Virtual Machine.
- Acts similar to an assembler.
- As soon as a java program is compiled, java bytecode is generated.
- Java bytecode is the machine code in the form of a **.class** file.
- With the help of java bytecode, we achieve platform independence in java.

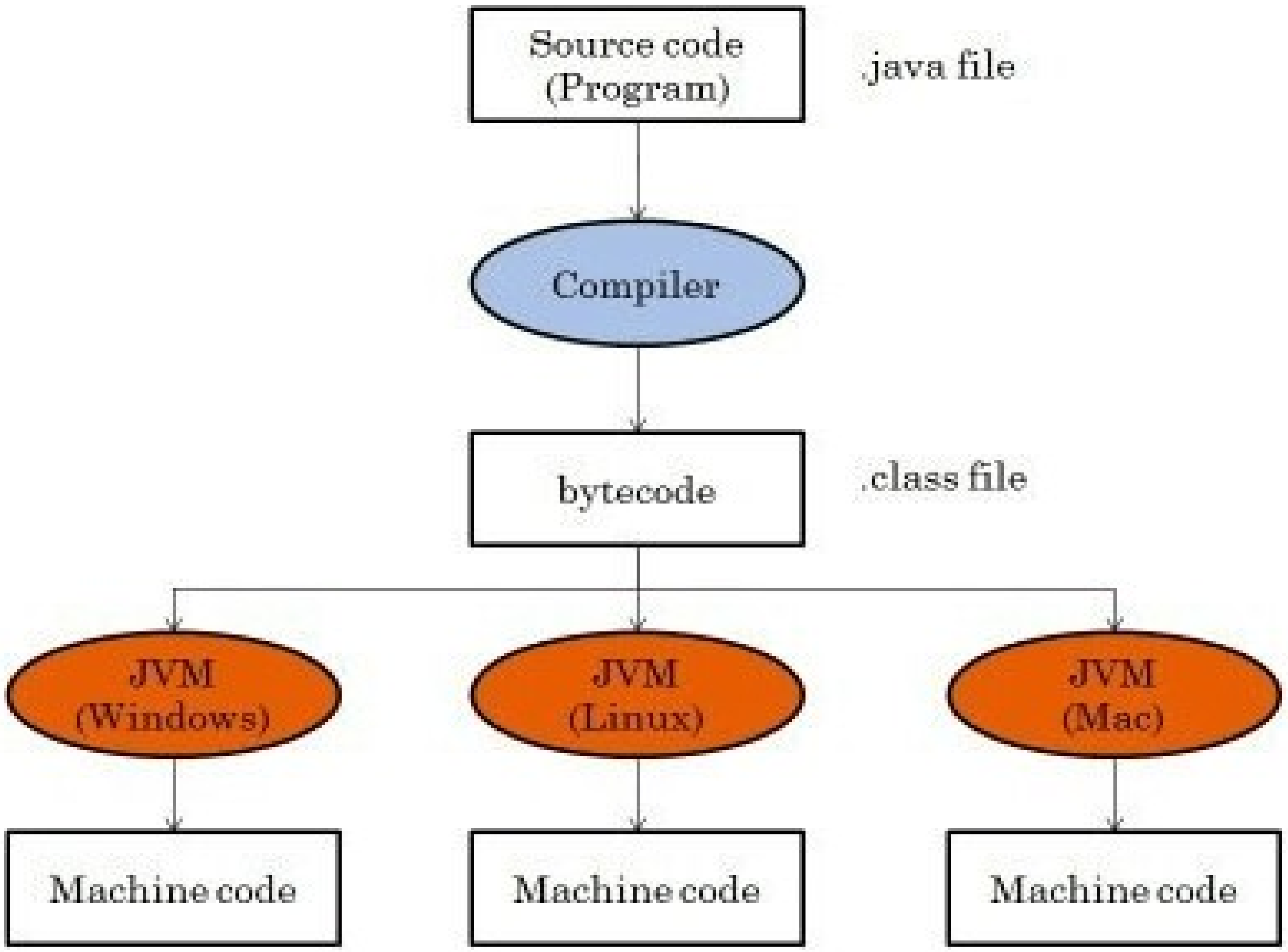
How Bytecode Works?

- Write a program in Java.
- Compiler compiles that program and a bytecode is generated for that piece of code.
- Bytecode generated is now run by the Java Virtual Machine.
- Need to have basic java installation on any platforms that we want to run our code on.
- Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources.

More About Java Bytecode

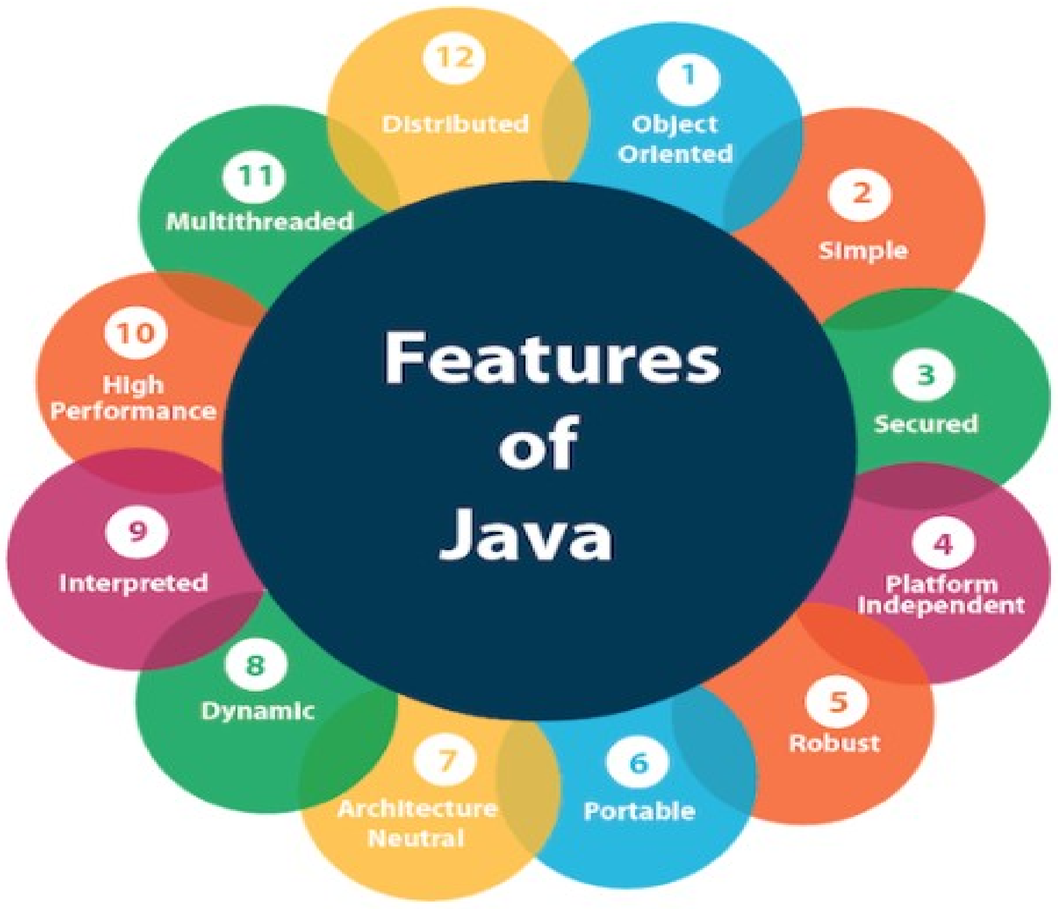
- Bytecode implementation makes Java a platform-independent language.
- The set of instructions for the JVM may differ from system to system, but all can interpret the bytecode.
- Bytecodes are non-runnable codes and rely on the availability of an interpreter to execute and thus the JVM comes into play.
- Bytecode is essentially the machine level language which runs on the Java Virtual Machine.
- Javac not only compiles the program but also generates the bytecode for the program.

How Bytecode Works?



Java Buzzwords

- Simple
- Object-Oriented
- Portable
- Platform independent
- Secured
- Robust
- Architecture neutral
- Interpreted
- High Performance
- Multithreaded
- Distributed
- Dynamic



Simple

- Java is very easy to learn, and its syntax is simple, clean and easy to understand.
  - Java syntax is based on C++
  - Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
  - There is no need to remove unreferenced objects because there is an **Automatic Garbage Collection** in Java.

Object-Oriented

- Java is an object-oriented programming language.
- Everything in Java is an object.
- Object-oriented means, we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Platform Independent

- Java is platform independent because it is different from other languages like C, C++ etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.
- A platform is the hardware or software environment in which a program runs.
- Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.

Secured

Java is **secured** because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine**
- **Classloader**
  - Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically.
  - It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Secured

Java is **secured** because:

- **Bytecode Verifier**
  - It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager**
  - It determines what resources a class can access such as reading and writing to the local disk.

Robust

Java is **robust** because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java.
- All these points make Java robust.

Architecture-neutral

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
- In C programming, **int** data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

- Java is **portable** because it facilitates you to carry the Java bytecode to any platform.

High Performance

- Java is faster than other traditional **interpreted** programming languages because Java bytecode is "close" to native code.
- It is still a little bit slower than a **compiled** language (e.g., C++).
- Java is an **interpreted** language that is why it is slower than **compiled** languages, e.g., C, C++, etc.

Distributed

- Java is distributed because it facilitates users to create distributed applications in Java.
- RMI and EJB are used for creating distributed applications.
- This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

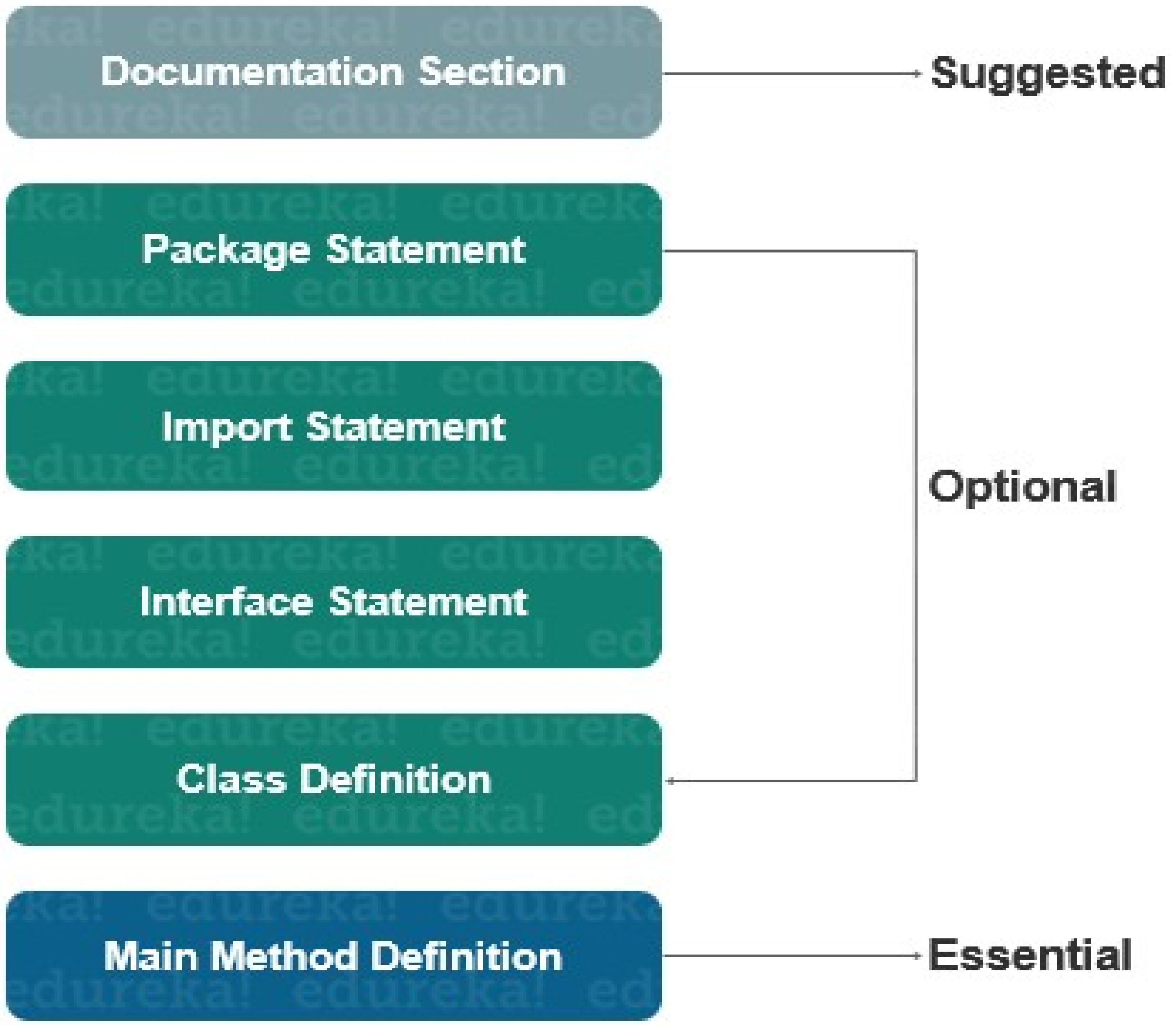
- A thread is like a separate program, executing concurrently.
- We can write Java programs that deal with many tasks at once by defining multiple threads.
- The main advantage of multi-threading is that it doesn't occupy memory for each thread.
- It shares a common memory area.
- Threads are important for multi-media, Web applications, etc.

Dynamic

- Java is a dynamic language.
- It supports dynamic loading of classes. It means classes are loaded on demand.
- Java supports dynamic compilation and automatic memory management (garbage collection).

Java Program Structure

*



*source: Edureka

Package Statement

- Java that allows you to declare your classes in a collection called Package.
- There can be only one package statement in a Java program and it has to be at the beginning of the code before any class or interface declaration.

```
package student;
```

- This statement declares that all the classes and interfaces defined in this source file are a part of the student package.
- Only one package can be declared in the source file.

Import Statement

- Many predefined classes are stored in packages in Java
- An import statement is used to refer to the classes stored in other packages.
- An import statement is always written after the package statement but it has to be before any class declaration.
- We can import a specific class or classes in an import statement.
- Take a look at the example to understand how import statement works in Java.

```
            import java.util.Date; //imports the date class
            import java.applet.*;  //imports all the classes from
the                                        java applet package
            import java.applet.Applet; // specific class Applet
```

Interface Section

- This section is used to specify an interface in Java.
- It is an optional section which is mainly used to implement multiple Inheritance in Java.
- An interface is a lot similar to a class in Java but it contains only constants and method declarations (abstract way).
- An interface cannot be instantiated but it can be implemented by classes or extended by other interfaces.

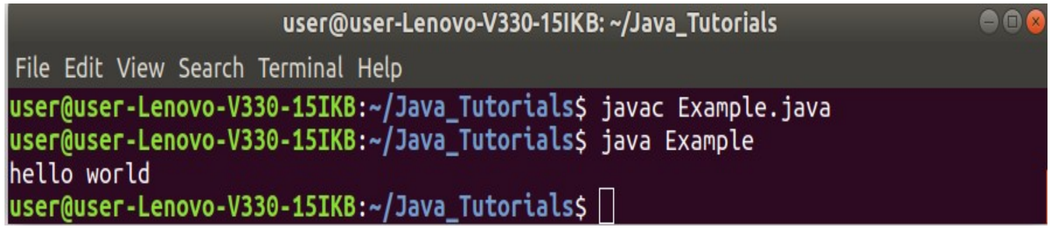
```
interface stack{
      void push(int item);
      void pop();
}
```

Class Definition

- Java program may contain several class definitions, classes are an essential part of any Java program.
- It defines the information about the user-defined classes in a program.
- A class is a collection of variables and methods that operate on the fields.
- Every program in Java will have at least one class with the main method.
- **main Method Class**
  - The main method is the method from where the execution actually starts and follows the order specified for the following statements.

Sample Program

```
public class Example{
    //main method declaration
    public static void main(String[] args){
        System.out.println("hello world");
    }
}
```


user@user-Lenovo-V330-15IKB: ~/Java_Tutorials
File Edit View Search Terminal Help

```
user@user-Lenovo-V330-15IKB:~/Java_Tutorials$ javac Example.java
user@user-Lenovo-V330-15IKB:~/Java_Tutorials$ java Example
hello world
user@user-Lenovo-V330-15IKB:~/Java_Tutorials$ 
```

- **public class Example**
  - This creates a class called **Example**.
  - You should make sure that the class name starts with a capital letter, and the public word means it is accessible from any other classes.

public static void main

- When the main method is declared **public**, it means that it can be used outside of this class as well.
- The word **static** means that we want to access a method without making objects.
- i.e, we call the main method without creating any objects.
- The main is declared as **void** because it does not return any value.

public static void main

- **String[] args**
  - It is an array where each element is a string, which is named as args.
  - If you run the Java code through a console, you can pass the input parameter.
  - The main() takes it as an input.

- Compiling Command: ***javac Example.java***
- Executing Command: ***java Example***

Comments

- The compiler ignores these comments during the time of execution and is used for improving the readability of the Java program.
- There are three types of comments that Java supports
  - Single line Comment
  - Multi-line Comment
  - Documentation Comment
- // a single line comment is declared like this
- /* a multi-line comment is declared like this
  and can have multiple lines as a comment */
- /** a documentation comment starts with a delimiter and
  ends with */

Lexical Tokens

-
- Java programs are a collection of
    - Whitespace
    - Identifiers
    - Literals
    - Comments
    - Separators
    - Keywords

Lexical Tokens

- **Whitespace:**
  - Java is a free-form language.
  - It means we do not need to follow any special indentation rules.
  - In Java, whitespace is a space, tab, or newline.

Lexical Tokens

- **Identifiers:**
  - Identifiers are used for class names, method names and variable names.
  - An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollor-sign characters.
  - **Eg:** AvgTemp, count, $calculate, s5, value_display
  - An identifier **must not** begin with a number because it leads to invalid identifier.

Lexical Tokens

- **Literals:**
  - A constant value in Java is created by using a literal representation.
  - A literal is allowed to use anywhere in the program.
  - Eg: Integer literal : 100

    Floating-point literal : 98.6

    Character literal : ‘s’

    String literal : “sample”

Lexical Tokens

- **Comments:**
  - The contents of a comment are ignored by the compiler.
  - A comment describes or explains the operation of the program to anyone who is reading its source code.
  - The comment describes the program.
  - In java, there are three types of comments.
  - They are single-line, multi-line and documentation comment.

Lexical Tokens

- **Separators:**
  - Separators are used to terminate statements.
  - In java, there are few characters are used as separators.
  - They are:
    - Parentheses ()
    - Braces {}
    - Brackets []
    - Semicolon ;
    - Period .
    - Comma ,

Lexical Tokens

- **Keywords:**
  - In java, a **keyword** has a predefined meaning in the language, because of this, programmers cannot use keywords as names for variables, methods, classes, or as any other identifier.
  - **main**
  - **boolean**
  - **break**
  - **byte, etc.**