**JAVA**

**Benefits of Java :**

* The ability to write once run it anywhere (WORA).
* The ability to develop Object Oriented Solutions.
* Programming in a familiar C based syntax.
* The availability of a rich set of libraries.

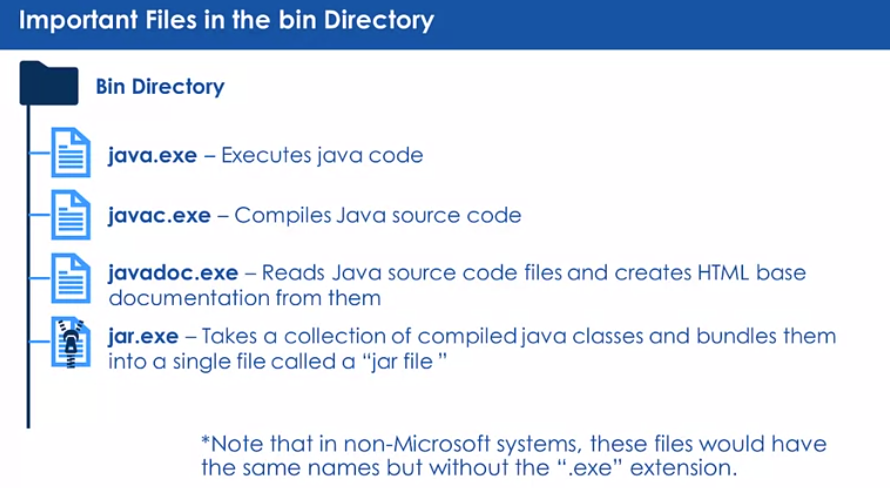
- You can have many files into directory just as you can have many classes in a package

- Java comes with a rich set of libraries, string manipulations, calendars, text, handling, XML parsing, and many other things.

* Built in levels of security.
* Support for internationalization .
* High-performance.
* The ability to use Java at no charge (free).
* A community that continues to evolve the rich Java Ecosystem.

\* Java code is compiled into an 'intermediate format' called architecture neutral bytecodes, and packages are Java's platform-neutral version of directories.

\* A specification of the API: Interfaces and Required Behavior is defined by the JCP (Java Community Process)



**Names in java** :

* Names are a sequence of letters, numbers, and underscores.
* Class names utilize Camel Case.
* Names are used to denote classes, objects, attributes, and methods.

**AutoBoxing** :

Is the automatic conversion the Java compiler makes between the primitive types and their corresponding object wrapper classes (ex: converting an int to an Integer).

**Ternary :**

Java ternary operator is the only conditional operator that takes three operands. It’s a one-liner replacement for the if-then-else statement and is used a lot in Java programming :

*Syntax***:**  variable = Expression1 ? Expression2: Expression3

*EXTRA :*

\* while (1) { } : NOT VALIDE, 1 is not a valid boolean value.

\* **Java switch** statement is a fall through statement that means it executes all statements if **break** keyword is not used.

**JAVA - OOP**

**Why use OO ?**

* For **managing complexity**. OO programming uses "encapsulation" to isolate complex sections of code.
* For **code Re-use.** In **OO** systems, code is created in such a way that it can be re-used by any program that needs it (removing redundancy).
* For managing change. OO design addresses this through ''functional separation" (encapsulation)

\* The fewer enhancements and couplings that are added, the more flexible your code.

**Design time :**

We can define the method name and the attribute's data type (optional) at design time.

**Objects :**

- Every object will have a reference value so that it can be uniquely identified.

- Objects(contain attributes & methods) are created within a program at runtime using a class as a template.

**JavaBean:**

A JavaBean is just a [standard](http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html). It is a regular Java class, except it follows certain conventions:

1. All properties are private (use [getters/setters](http://en.wikipedia.org/wiki/Mutator_method))
2. A public [no-argument constructor](http://en.wikipedia.org/wiki/Nullary_constructor)
3. Implements [Serializable](http://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html).

That's it. It's just a convention. Lots of libraries depend on it though. With respect to Serializable

\* A **JavaBean property** is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

\* The **Serializable**interface is present in **java.io** package. It is a [marker interface](https://www.geeksforgeeks.org/marker-interface-java/). A Marker Interface does not have any methods and fields.

***Serialization is a mechanism of converting the state of an object into a byte stream. Serialization is done using***[*ObjectOutputStream*](https://www.geeksforgeeks.org/java-io-objectoutputstream-class-java-set-1/)***. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object. Deserialization is done using***[*ObjectInputStream*](https://www.geeksforgeeks.org/java-io-objectinputstream-class-java-set-2/)***. Thus it can be used to make an eligible for saving its state into a file.***

\* The **method signature** in java is defined as the structure of the method that is designed by the programmer. The method signature is the combination of the method name and the parameter list.