**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.

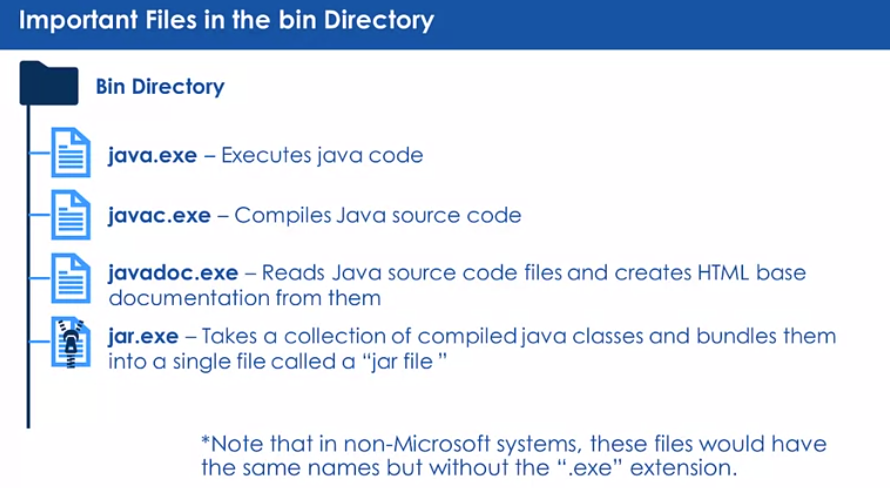
- You can have many files into directory just as you can have many classes in a package

- 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.

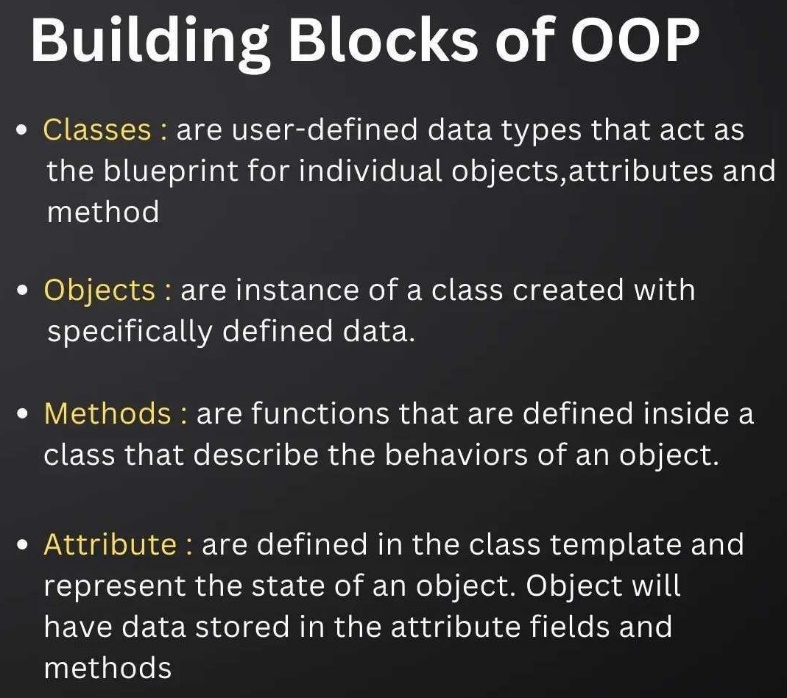
**JavaBean:**

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).

**Package**:

A package is a namespace that mainly contains classes and interfaces. For instance, the standard class **ArrayList** is in the package **java.util**. For this class, **java.util.ArrayList** is called its fully qualified name because this syntax has no ambiguity.



\* 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.

**Java class path:**

The Java class path tells a Java compiler or Java Virtual Machine (JVM) where to look for Java classes and libraries needed to compile or run Java programs. There are two ways to set the class path: with an environment variable or with a command-line option of the JVM.

**STATIC**:

The **static keyword** in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

**Nested Classes**:

Nested classes are divided into two categories: non-static and static. Non-static nested classes are called inner classes. Nested classes that are declared static are called static nested classes.

class OuterClass {

...

class InnerClass {

...

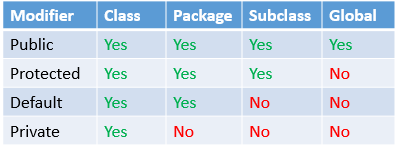
}

static class StaticNestedClass {

...

}

}



\* **Objects** are stored in memory and managed by the JVM in a reference table.

\* All **Objects** of the same type share the methods in memory but have unique data.

**JAR vs WAR:**

JAR files allow us to package multiple files in order to use it as a library, plugin, or any kind of application. On the other hand, **WAR** files are used only for web applications.

\*Java provides a program that collects all class files in one place called a JAR file.

**Java String**

In [**Java**](https://www.javatpoint.com/java-tutorial), string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java) of characters works same as Java string.  **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* [interfaces](https://www.javatpoint.com/interface-in-java).



## **CharSequence Interface**

The CharSequence interface is used to represent the sequence of characters. String, [StringBuffer](https://www.javatpoint.com/StringBuffer-class) and [StringBuilder](https://www.javatpoint.com/StringBuilder-class) classes implement it. It means, we can create strings in Java by using these three classes.

String is **immutable** whereas StringBuffer and StringBuilder are **mutable** classes. StringBuffer is thread-safe and synchronized whereas StringBuilder is not. That's why StringBuilder is faster than StringBuffer.

\***thread safe**. It means two threads can't call the methods of StringBuffer simultaneously.

There are two ways to create String object: By string literal or By new keyword :

Java String literal is created by using double quotes. Each time you create a string literal, the JVM checks the "**string constant pool**" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

***String s1="Welcome";***

***String s2="Welcome"; //It doesn't create a new instance***

***In Java, string objects are immutable, which means once it is created, you cannot change it. So when we concatenate one string with another, a new string is created, and the older one is marked for the garbage collector. Let's say we need to concatenate a million strings. Then, we are creating 1 million extra strings which will eventually be garbage collected.***

TODO : garbage collector && synchronization

<https://www.geeksforgeeks.org/garbage-collection-java/#:~:text=Garbage%20collection%20in%20Java%20is,memory%20dedicated%20to%20the%20program>.

**Advance OOPs**

**Inheritance:**

Inheritance allows us to include the attributes and methods of one class in another related, in java:

1. Single inheritance (can inherit only one class).
2. Multiple inheritance of interfaces.

**Polymorphism:**

In java, it means that two or more classes may have the same method name and signature but different implementations of the code.

**Association:**

Is a relationship between classes. Association allows objects of one class to send messages to other objects.

**Abstract Class:**

An abstract class is a class that is used as a template for other classes. And cannot be instantiated. Also A partially implemented class.

**Verbs VS Nouns:**

To describe object oriented programming, objects are often referred to as “nouns” and actions that determine their behavior as “verbs”. These verbs are traditionally implemented as methods, i.e. systematically coupled to the object that 'performs' the action

**Aggregation VS Composition:**

Association in java is one of the types of relations between classes. It has two forms Aggregation (HAS-A) and Composition(Belongs-to). Aggregation is a relatively weak association, whereas Composition is a strong association. Composition can be called a more restricted form of Aggregation

\*Is-A relationship : inheritance.

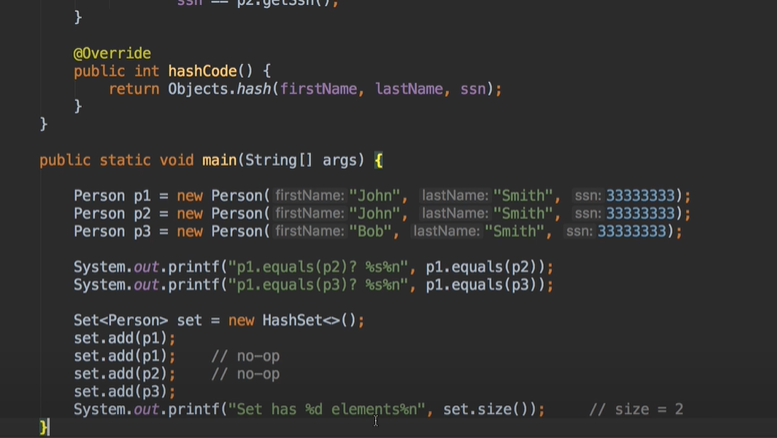
\*Has-A relationship: Aggregation/Composition

\* Use verbs when naming methods and nouns when naming classes.

**Inheritance**

**Equals and hashcode:**

An object hash code value can change in multiple executions of the same application.**If two objects are equal according to equals() method, then their hash code must be same**. If two objects are unequal according to equals() method, their hash code are not required to be different.



**Overloading and Overriding:**

**Overriding** occurs when the method signature is the same in the superclass and the child class. **Overloading** occurs when two or more methods in the same class have the same name but different parameters.

\* Car car; // the car variable is a reference with undefined value.

\* All Java classes automatically directly or indirectly extend ***java.lang.Object***

**Abstract Class**

**Where to use :**

An object hash code value can change in multiple

**Casting :**

An object hash code value can change in multiple

\* **Upcasting** is casting to a supertype, while **downcasting** is casting to a subtype. **Upcasting** is always allowed, but **downcasting** involves a type check and can throw a ClassCastException.

**Interfaces**

Interfaces vs abstract classes, default and static methods in java 8, functional interfaces, lambdas…

**Polymorphism**

What is it and Why, coding examples…

**JAVA CLASS LIBRARY**

**Generics**

Importance, generic with subtypes without willcards, generic methods, type erasure…

**Collections**

Importance, framework, set, iterator, sorting and comparing, lists, maps, streams…

**Exception**

Exception handling, checked/unchecked/standard exceptions, catching/cleaning, try/throwing …

**File I/O**

Property files, Java.io streams, writing text files, input output and Exception…

**Enums**

What and why, types, implementing enums, enumMap…

**Annotations**

Why, declaring and using annotations, meta-annotations, developing annotations…