Features of java



Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

* Java syntax is based on C++ (so easier for programmers to learn it after 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

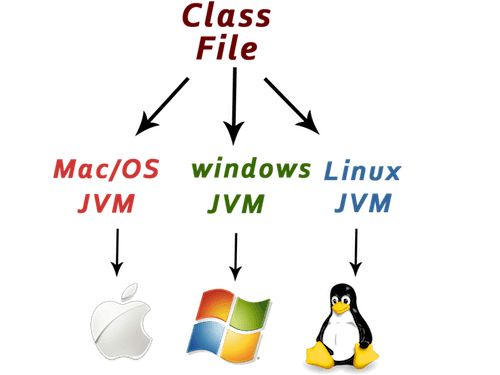
Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. Class
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

Platform Independent



Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), 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.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

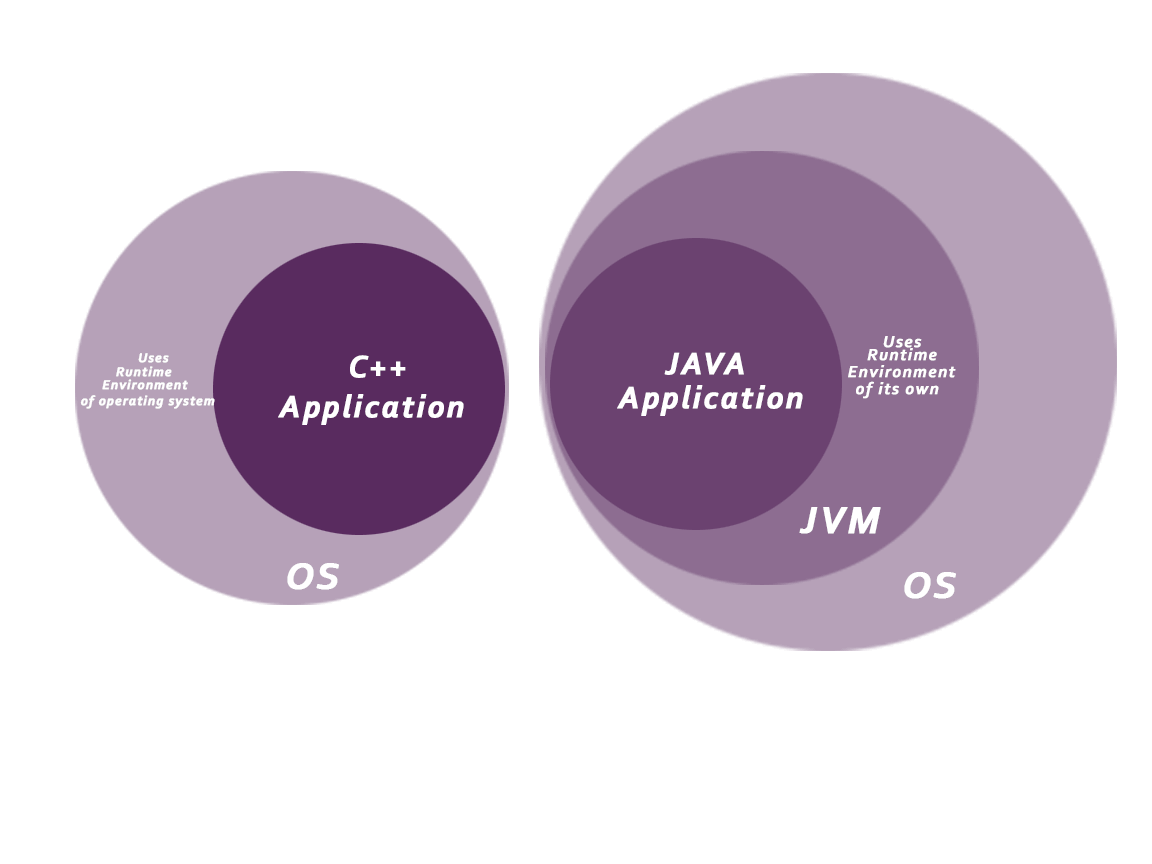
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

* **No explicit pointer**
* **Java Programs run inside a virtual machine sandbox**



* **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.
* **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.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

Robust simply means strong. 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. It doesn't require any implementation.

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

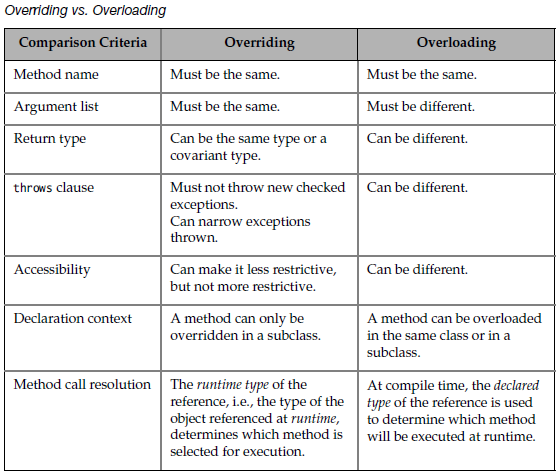
A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media,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

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

Overloding vs overriding



String method

### *compareTo(String anotherString)*

Often, it is not enough to simply know whether two strings are identical. For sorting applications, you need to know which is less than, equal to, or greater than the next. A string is less than another if it comes before the other in dictionary order.  
  
A string is greater than another if it comes after the other in dictionary order. The method*compareTo()*  serves this purpose. It is specified by the Comparable interface, which String implements. It has this general form:

int compareTo(String str)

### *compareToIgnoreCase(String str)*

Compares two strings lexicographically, ignoring case differences. This method returns an integer whose sign is that of calling compareTo with normalized versions of the strings where case differences have been eliminated by calling  *Character.toLowerCase(Character.toUpperCase(character))*on each character.

### *concat(String str)*

Concatenates the specified string to the end of this string.

This method creates a new object that contains the invoking string with the contents of str appended to the end.  *concat()*  performs the same function as +.

Constructor overloading

Constructor [overloading in Java](https://www.javatpoint.com/method-overloading-in-java) is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

1. //Java program to overload constructors
2. **class** Student5{
3. **int** id;
4. String name;
5. **int** age;
6. //creating two arg constructor
7. Student5(**int** i,String n){
8. id = i;
9. name = n;
10. }
11. //creating three arg constructor
12. Student5(**int** i,String n,**int** a){
13. id = i;
14. name = n;
15. age=a;
16. }
17. **void** display(){System.out.println(id+" "+name+" "+age);}
19. **public** **static** **void** main(String args[]){
20. Student5 s1 = **new** Student5(111,"Karan");
21. Student5 s2 = **new** Student5(222,"Aryan",25);
22. s1.display();
23. s2.display();
24. }
25. }

Inheritance

* It helps us to inherit attributes and methods from one class to another. Inheritance is achieved using the **extends** keyword
* Reusability is an important concept in OOP, So when a new class is created it must be able to reuse the properties of the existing class
* The class from where the inheritance starts is called the Base class/Super class

SINGLE INHERITANCE

* In single inheritance the structure will have only a single parent class and a single child class
* The Derived Class is authorized to access the property of the Base class

MULTILEVEL INHERITANCE

* Multi level inheritance has the standard structure of a single inheritance by having a single parent class but consists of one or more than one intermediate child classes
* The derived class as well as the intermediate class may access the properties of the upper level classes

PROGRAM TO IMPLEMENT STATIC VARIABLES AND STATIC METHOD

* **Static Variable:**
* In java when a variable declared as static, it is called a class variable and all objects (instances) share the same variable.
* **Static Method:**
* A static method always accesses static data and it also belongs to class not instance. Hence the static method is called with the class name not object name. It cannot be referred to like this or super keyword.
* **Program:**
* // Java program to demonstrate example of
* // static variable and static method
* **import** java.util.**\***;
* **class** Item {
* **private** **String** itemName;
* **private** int quantity;
* **private** **static** int cnt = 0; //variable to count
* **public** void getItem() {
* **Scanner** sc = **new** **Scanner**(**System**.in);
* **System**.out.print("Enter item name: ");
* itemName = sc.next();
* **System**.out.print("Enter item quantity: ");
* quantity = sc.nextInt();
* //increment counter
* cnt++;
* }
* **public** void showItem() {
* **System**.out.println("Item Name: " + itemName + "\tQuantity: " + quantity);
* }
* **public** **static** int getCounter() {
* **return** cnt;
* }
* }
* **public** **class** StaticVar {
* **public** **static** void main(**String**[] s) {
* **try** {
* Item I1 = **new** Item();
* Item I2 = **new** Item();
* Item I3 = **new** Item();
* I1.getItem();
* I2.getItem();
* I3.getItem();
* I1.showItem();
* I2.showItem();
* I3.showItem();
* **System**.out.println("Total object created (total items are): " + Item.getCounter());
* } **catch** (**Exception** e) {
* **System**.out.println(e.toString());
* }
* }
* }

PACKAGE

## **Java Packages & API**

A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

* Built-in Packages (packages from the Java API)
* User-defined Packages (create your own packages)

**Creating a Package**

To create a package, follow the steps given below:

* Choose a package name according to the naming convention.
* Write the package name at the top of every source file (classes, interface, enumeration, and annotations).
* Remember that there must be only one package statement in each source file.

## **Importing a Package**

If we want to use a package in [Java program](https://www.javatpoint.com/java-programs) it is necessary to import that package at the top of the program by using the import keyword before the package name.

**Syntax:**

1. **import** packageName;