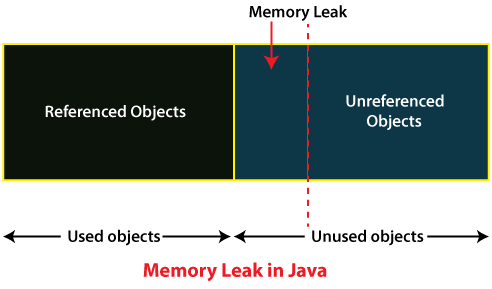
# Memory Leak in Java

**Java** provides out-of-box **memory management**. When we create an object using the new keyword, the JVM automatically allocates memory for that object. If the object is no longer is used by the application, the **garbage collector** automatically removes that object and free up space for other applications. Therefore, the programmer need not to manage memory manually like other procedural programming languages (C, and C++). Nevertheless, there is a chance of **memory leak** in a Java application. In this section, we will understand **what is a memory leak in Java**, its causes, detect and fixing of **memory leaks**.

## What is memory leak in Java?

In [Java](https://www.javatpoint.com/java-tutorial), the **memory leak** is a situation when the garbage collector does not recognize the unused objects and they remain in the memory indefinitely that reduces the amount of memory allocated to the application. Because the unused objects still being referenced that may lead to **OutOfMemoryError.** It also affects the reliability of the application. The following figure represents the memory leak.



#### Note: We can specify the initial and maximum heap size for the application. There are the two parameters (options) for setting up the heap size:

* -Xms<size>m
* -Xmx<size>m

Where **m** denotes the size in MB.

### Symptoms of Memory Leak

If any Java application suffers from a memory leak, you notify the following things:

* The performance of the application consistently decreases.
* The usage of memory increases during the life span of an application.

### Causes of Memory Leaks

There are the following causes of memory leaks in Java:

* **Using Unwanted Object Reference:** These are the object references that are no longer needed. The garbage collector is failed to reclaim the memory because another object still refers to that unwanted object.
* **Using Long-live Static Objects:** Using static objects also leads to a memory leak. Because they live in the memory till the application's life span.
* **Failure to Clean-up Native System Resources:** Native system resources allocated by a function external to Java. It is written in C and C++. JNI APIs are used to embed native libraries in the Java code.
* **Bugs in the Third-party Libraries:** Bugs in AWT and Java Swing packages are another cause of memory leak.

## Preventing Memory Leak

While writing code, remember the following points that prevent the memory leak in Java.

* Do not create unnecessary objects.
* Avoid String Concatenation.
* Use String Builder.
* Do not store a massive amount of data in the session.
* Time out the session when no longer used.
* Do not use the System.gc() method.
* Avoid the use of static objects. Because they live for the entire life of the application, by default. So, it is better to set the reference to null, explicitly.
* Always close the ResultSet, Statements, and Connection objects in the finally block.

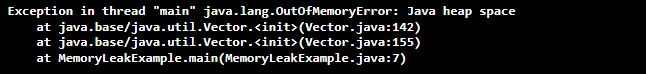
## Creating Memory Leak

Let's create a simple Java program that creates a memory leak.

**MemoryLeakExample.java**

1. **import** java.util.Vector;
2. **public** **class** MemoryLeakExample
3. {
4. **public** **static** **void** main(String[] args)
5. {
6. Vector v1 = **new** Vector(314567);
7. Vector v2 = **new** Vector(876543987);
8. System.out.println("There is no memory leak in this program.");
9. }
10. }

**Output**

****

In the above program, we have created two Vector objects and passed large numbers to them. When we run the above program, it shows **java.lang.OutOfMemoryError.** Because it does not occupy space in the memory. If the program prints the statement **There is no memory leak in this program,** it ensures that the program runs successfully.

### Detecting Memory Leak

Detecting memory leaks is a difficult task. To simplify the task, there are many tools available that perform static analysis and detect memory leaks:

* JProbe
* AppPerfect
* Visual VM
* Jprofiler
* YourKit
* GCeasy
* JRockit

## Fixing Memory Leak

There are the following solutions to the memory leak problem:

* **Using JVM Tools:** There are many tools available that optimizes the code and show the memory status.
* **Using Heap Dump:** It is a technique that is the solution to the memory leak problem. It is a snapshot of all objects that reside in the memory at a certain time. It also optimizes memory usage in a Java application. It is stored in binary format in **hprof**
* **Using Eclipse Memory Leak Warnings:** If you are using the Eclipse framework to develop a Java application, eclipse regularly shows the waring and errors whenever it encounters any causes of memory leak.