Difference between JDK, JRE and JVM

1. [Brief summary of JVM](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm)
2. [Java Runtime Environment (JRE)](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm#jre)
3. [Java Development Kit (JDK)](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm#jdk)

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM.

# JVM (Java Virtual Machine)

1. [Java Virtual Machine](https://www.javatpoint.com/internal-details-of-jvm)
2. [Internal Architecture of JVM](https://www.javatpoint.com/internal-details-of-jvm#jvminternalarch)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

### What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

### What it does

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

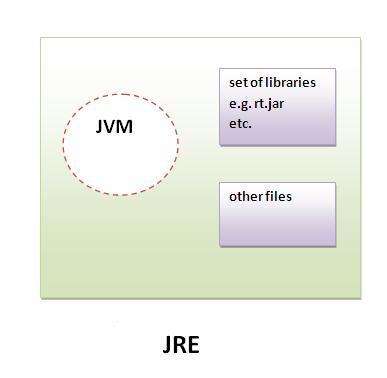
The JVM performs following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

### JRE

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



### JDK

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.



# Java Heap Space vs Stack – Memory Allocation in Java

July 17, 2017 by [Pankaj](https://www.journaldev.com/author/pankaj)

Sometime back I wrote a couple of posts about [Java Garbage Collection](https://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java) and [Java is Pass by Value](https://www.journaldev.com/3884/java-is-pass-by-value-and-not-pass-by-reference). After that I got a lot of emails to explain about **Java Heap Space**, **Java Stack Memory**, **Memory Allocation in Java** and what are the differences between them.

You will see a lot of reference to Heap and Stack memory in Java, Java EE books and tutorials but hardly complete explanation of what is heap and stack memory in terms of a program.

## Java Heap Space

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space.

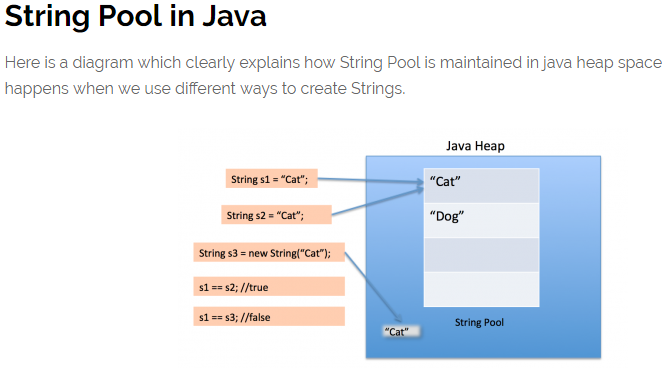
Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

### Java Stack Memory

Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method.

Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.

As soon as method ends, the block becomes unused and become available for next method.  
Stack memory size is very less compared to Heap memory.



### Heap and Stack Memory in Java Program

Let’s understand the Heap and Stack memory usage with a simple program.

package com.journaldev.test;

public class Memory {

public static void main(String[] args) { // Line 1

int i=1; // Line 2

Object obj = new Object(); // Line 3

Memory mem = new Memory(); // Line 4

mem.foo(obj); // Line 5

} // Line 9

private void foo(Object param) { // Line 6

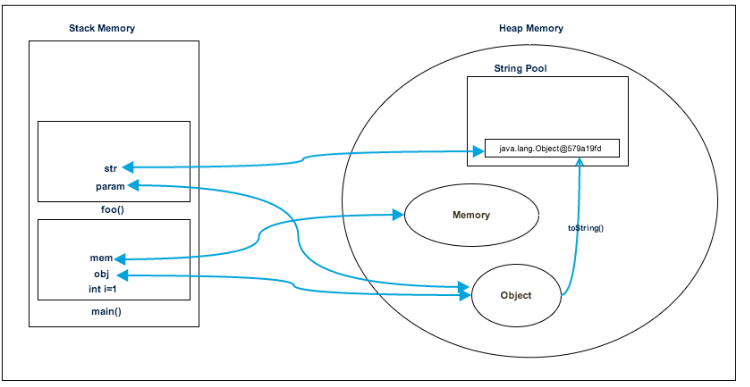
String str = param.toString(); //// Line 7

System.out.println(str);

} // Line 8

}

Below image shows the Stack and Heap memory with reference to above program and how they are being used to store primitive, Objects and reference variables.



Let’s go through the steps of execution of the program.

* As soon as we run the program, it loads all the Runtime classes into the Heap space. When main() method is found at line 1, Java Runtime creates stack memory to be used by main() method thread.
* We are creating primitive local variable at line 2, so it’s created and stored in the stack memory of main() method.
* Since we are creating an Object in line 3, it’s created in Heap memory and stack memory contains the reference for it. Similar process occurs when we create Memory object in line 4.
* Now when we call foo() method in line 5, a block in the top of the stack is created to be used by foo() method. Since Java is pass by value, a new reference to Object is created in the foo() stack block in line 6.
* A string is created in line 7, it goes in the [String Pool](https://www.journaldev.com/797/what-is-java-string-pool) in the heap space and a reference is created in the foo() stack space for it.
* foo() method is terminated in line 8, at this time memory block allocated for foo() in stack becomes free.
* In line 9, main() method terminates and the stack memory created for main() method is destroyed. Also the program ends at this line, hence Java Runtime frees all the memory and end the execution of the program.

## Difference between Java Heap Space and Stack Memory

Based on the above explanations, we can easily conclude following differences between Heap and Stack memory.

1. Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
2. Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.
3. Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.
4. Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at [Java Garbage Collection](https://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java).
5. Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.
6. We can use **-Xms** and **-Xmx** JVM option to define the startup size and maximum size of heap memory. We can use **-Xss** to define the stack memory size.
7. When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.
8. Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

That’s all for **Java Heap Space vs Stack Memory** in terms of java application, I hope it will clear your doubts regarding memory allocation when any java program is executed.

### What is finally and finalize in java?

finally block is used with try-catch to put the code that you want to get executed always, even if any exception is thrown by the try-catch block. finally block is mostly used to release resources created in the try block.

finalize() is a special method in Object class that we can override in our classes. This method get’s called by garbage collector when the object is getting garbage collected. This method is usually overridden to release system resources when object is garbage collected.

### What are different types of classloaders?

There are three types of built-in Class Loaders in Java:

* 1. Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.
  2. Extensions Class Loader – It loads classes from the JDK extensions directory, usually $JAVA\_HOME/lib/ext directory.
  3. System Class Loader – It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.

**What is rt.jar?**

rt.jar stands for runtime JAR and contains the bootstrap classes, I mean all the classes from Core Java API.

The rt.jar is where all the Java  packages reside. For example, if a class file need to refer a class from java.util.concurrent package e.g. [ConcurrentHashMap](http://javarevisited.blogspot.sg/2013/02/concurrenthashmap-in-java-example-tutorial-working.html" \t "_blank), then the JVM will look for it inside the rt.jar, thus enabling it to run correctly.

