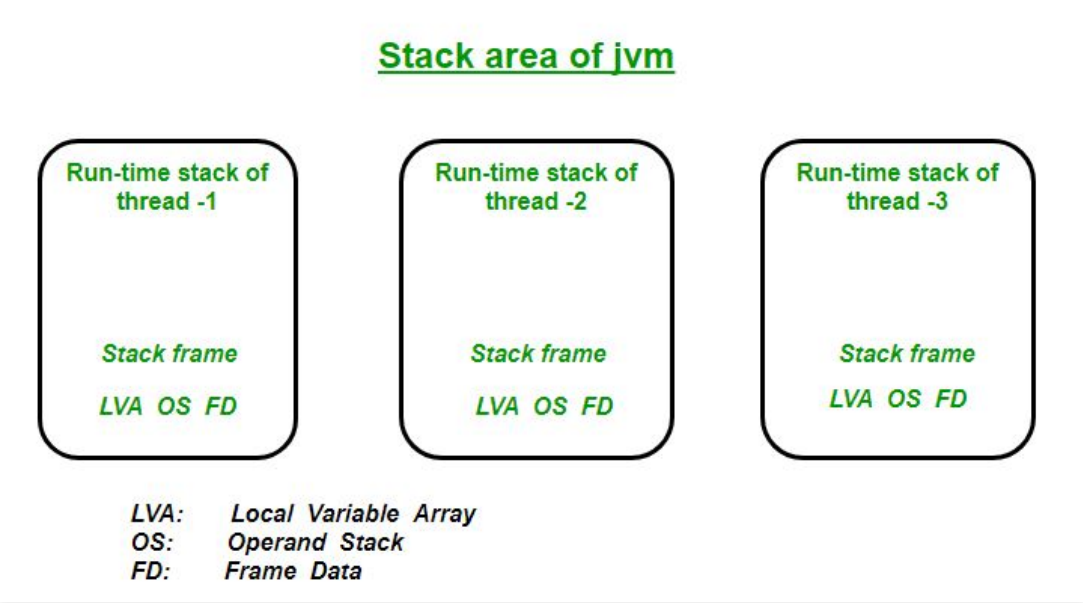
Java Virtual Machine (JVM) Stack Area

Difficulty Level : Medium Last Updated : 20 Jun, 2021

For every thread, JVM creates a separate stack at the time of thread creation. The memory for a Java Virtual Machine stack does not need to be contiguous. The Java virtual machine only performs two operations directly on Java stacks: it pushes and pops frames. And stack for a particular thread may be termed as **Run – Time Stack**. Every method call performed by that thread is stored in the corresponding run-time stack including parameters, local variables, intermediate computations, and other data. After completing a method, the corresponding entry from the stack is removed. After completing all method calls the stack becomes empty and that empty stack is destroyed by the JVM just before terminating the thread. The data stored in the stack is available for the corresponding thread and not available to the remaining threads. Hence we can say local data thread-safe. Each entry in the stack is called **Stack Frame** or **Activation Record**.



Stack Frame Structure

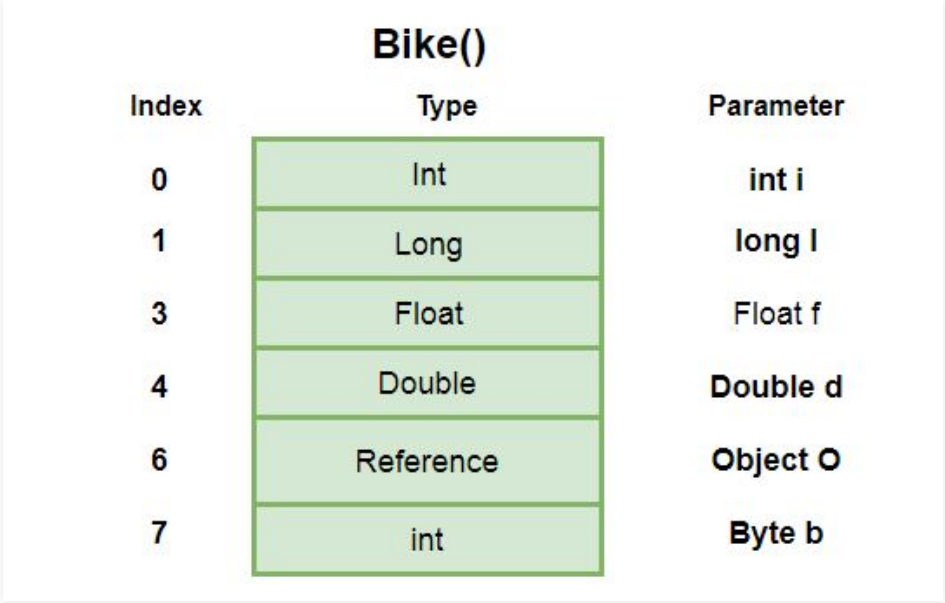
The stack frame basically consists of **three** parts: **Local Variable Array, Operand Stack & Frame Data**. When JVM invokes a Java method, first it checks the class data to determine the number of words *(size of the local variable array and operand stack, which* is *measured in words for each individual method)* required by the method in the local variables array and operand stack. It creates a stack frame of the proper size for invoked method and pushes it onto the Java stack.

1. Local Variable Array (LVA):

- The local variables part of the stack frame is organized as a zero-based array of words.
- It contains all parameters and local variables of the method.
- Each slot or entry in the array is of 4 Bytes.
- Values of type int, float, and reference occupy 1 entry or slot in the array i.e. 4 bytes.
- Values of double and long occupy 2 consecutive entries in the array i.e. 8 bytes total.
- **Byte, short**, **and char values will be converted to int type before storing** and occupy 1 slot i.e. 4 Bytes.
- But the way of storing Boolean values is varied from JVM to JVM. But most of the JVM gives 1 slot for Boolean values in the local variable array.
- The parameters are placed into the local variable array first, in the order in which they are declared.
- **For Example:** Let us consider a class Example having a method **bike()** then the local variable array will be as shown in the below diagram:

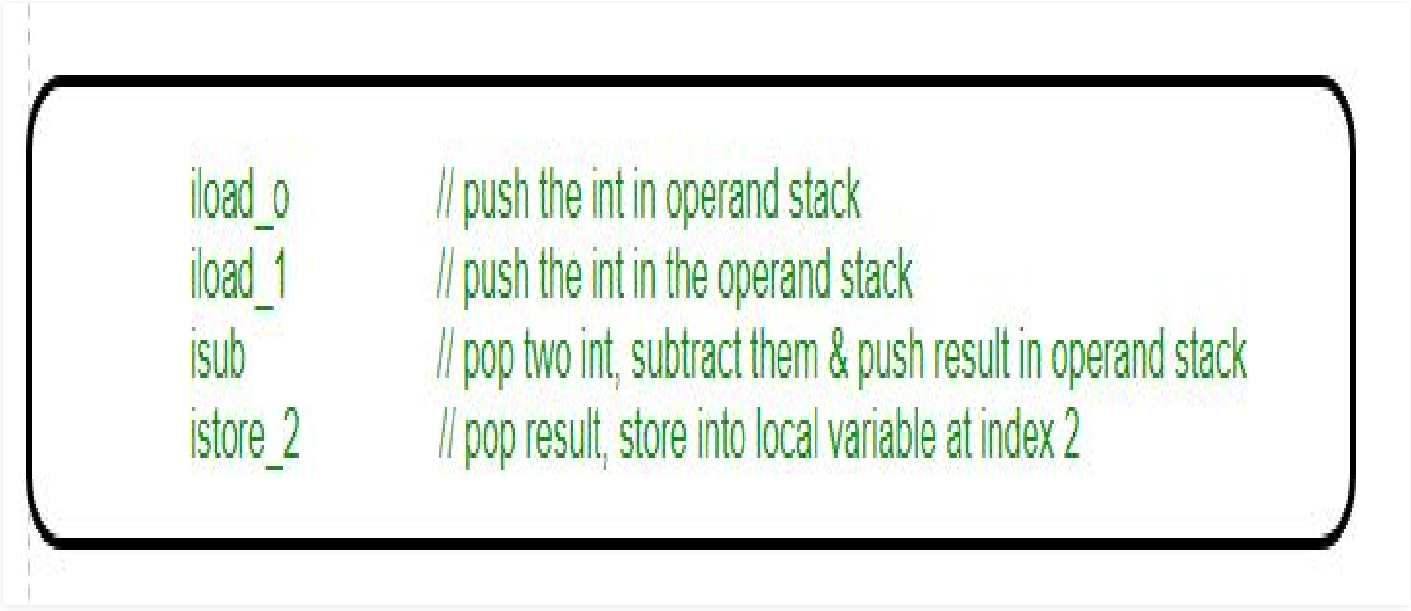
```
// Class Declaration
class Example
{
  public void bike(int i, long l, float f,
                double d, Object o, byte b)
  {

  }
}
```

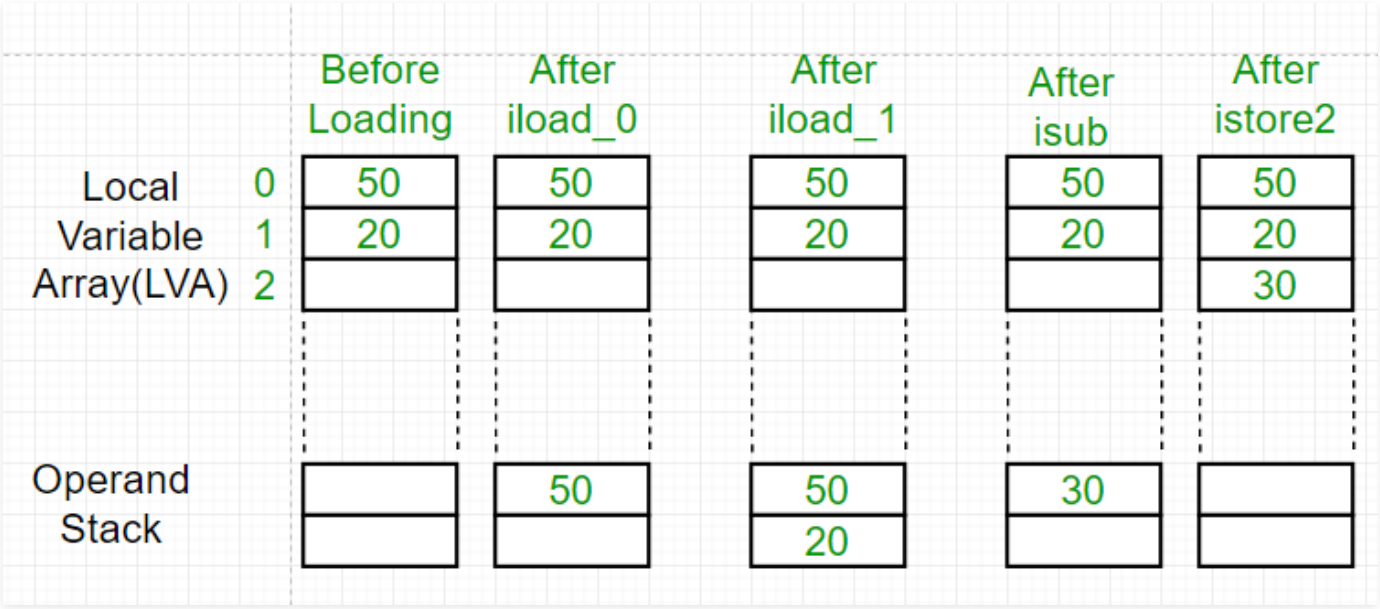


2. Operand Stack (OS):

- JVM uses operand stack as workspace like rough work or we can say for storing intermediate calculation's result.
- The operand stack is organized as an array of words like a local variable array. But this is not accessed by using an index like local variable array rather it is accessed by some instructions that can push the value to the operand stack and some instructions that can pop values from the operand stack and some instructions that can perform required operations.
- **For Example:** Here is how a JVM will use this below code that would subtract two local variables that contain two ints and store the int result in a third local variable:



- So here first two instructions *iload_0* and *iload_1* will push the values in the operand stack from a local variable array. And instruction *isub* will subtract these two values and store the result back to the operand stack and after *istore_2* the result will pop out from the operand stack and will store into a local variable array at position 2.



3. Frame Data (FD):

- It contains all symbolic references (*constant pool resolution)* and normal method returns related to that particular method.
- It also contains a reference to the Exception table which provides the corresponding catch block information in the case of exceptions.