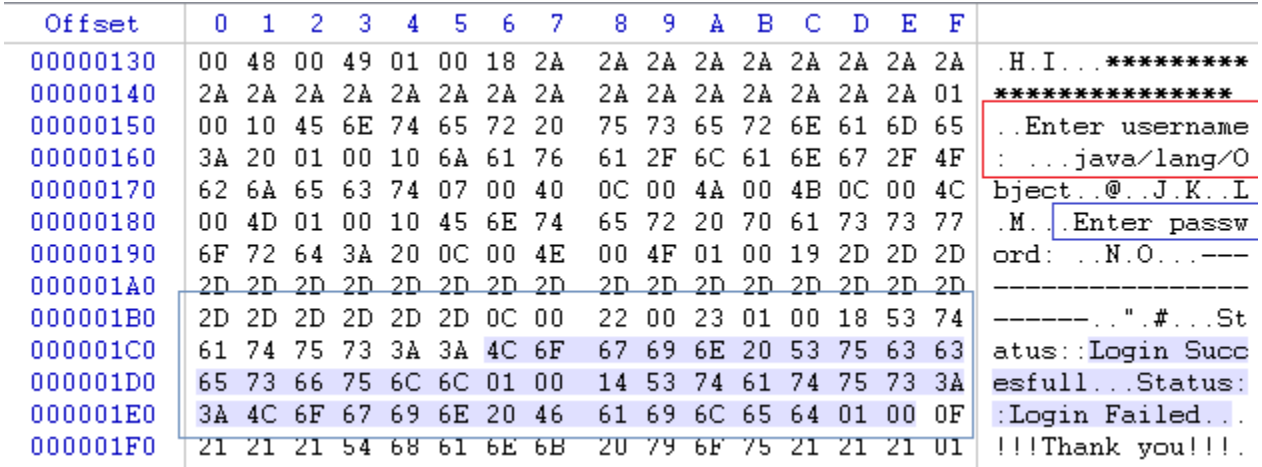
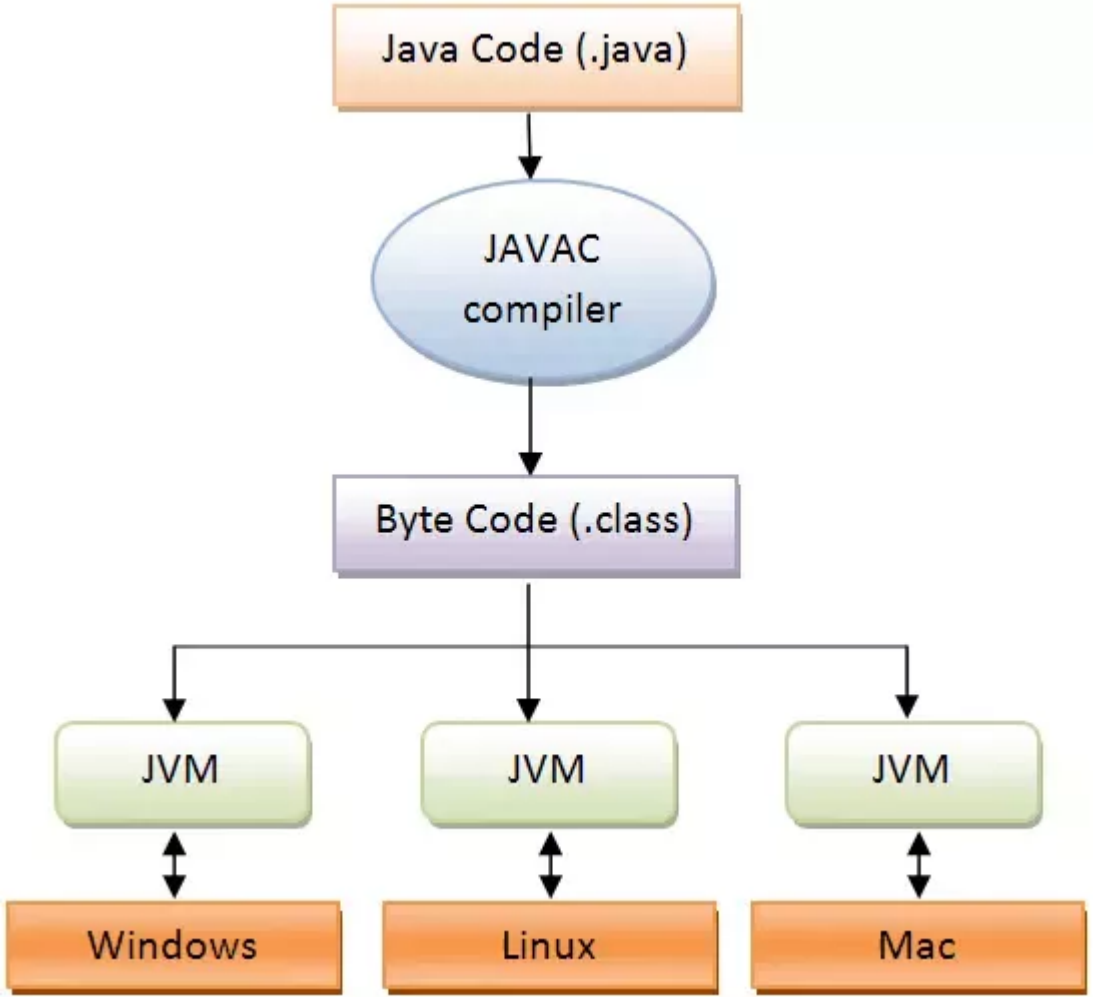
Bytecode & JVM



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000130	00	48	00	49	01	00	18	2A	2A	2A	2A	2A	2A	2A	2A	2A	.H.I...**********
00000140	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	01	**********
00000150	00	10	45	6E	74	65	72	20	75	73	65	72	6E	61	6D	65	..Enter username
00000160	3A	20	01	00	10	6A	61	76	61	2F	6C	61	6E	67	2F	4F	: ...java/lang/O
00000170	62	6A	65	63	74	07	00	40	0C	00	4A	00	4B	0C	00	4C	bject..@..J.K..L
00000180	00	4D	01	00	10	45	6E	74	65	72	20	70	61	73	73	77	.M...Enter passw
00000190	6F	72	64	3A	20	0C	00	4E	00	4F	01	00	19	2D	2D	2D	ord: ..N.O...---
000001A0	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	----------------
000001B0	2D	2D	2D	2D	2D	2D	0C	00	22	00	23	01	00	18	53	74	------..".#...St
000001C0	61	74	75	73	3A	3A	4C	6F	67	69	6E	20	53	75	63	63	atus::Login Succ
000001D0	65	73	66	75	6C	6C	01	00	14	53	74	61	74	75	73	3A	esfull...Status:
000001E0	3A	4C	6F	67	69	6E	20	46	61	69	6C	65	64	01	00	0F	:Login Failed...
000001F0	21	21	21	54	68	61	6E	6B	20	79	6F	75	21	21	21	01	!!!Thank you!!!.

JVM

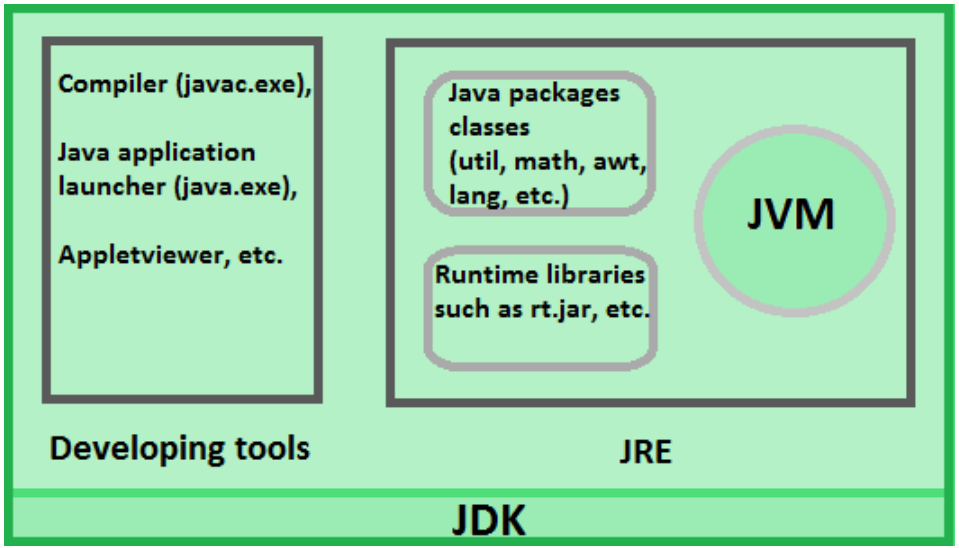
JVM = Implementation of:
Java
Virtual
Machine
Specification



For each platform specific JVM must be created based on Java Virtual Machine Specification *“interface”*.

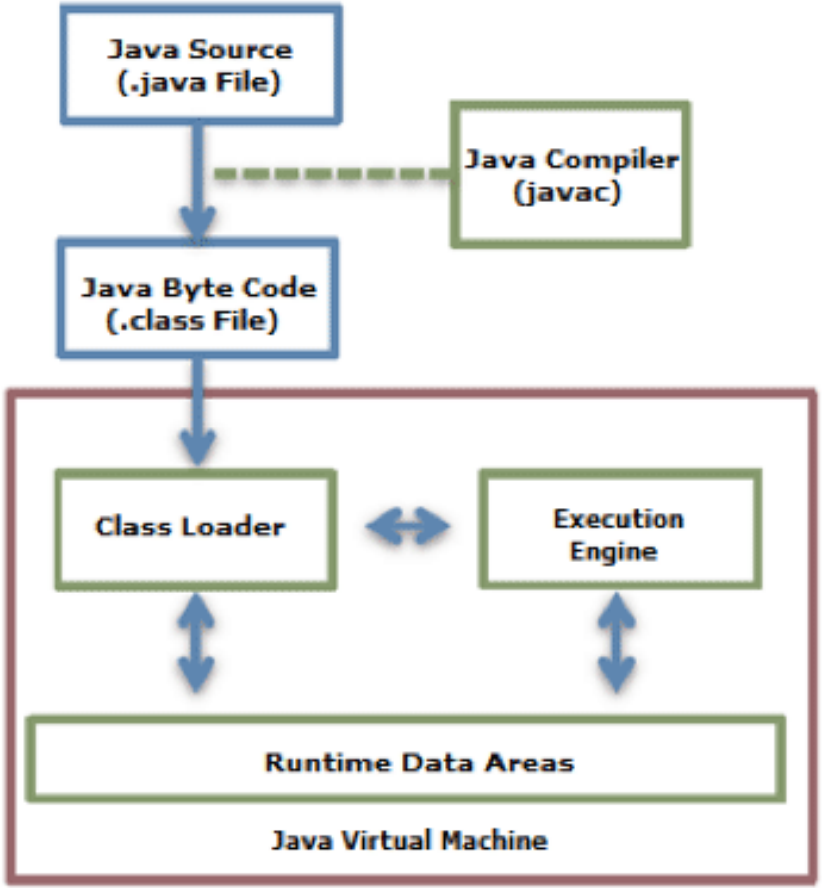
What is bytecode?

- Result of the compilation of a Java program, an **intermediate** representation of that program which is machine independent.
- Bytecode is executable on the platform where JRE is installed
- Bytecode is run by JVM.



JRE = Java API + JVM

http://www.letustweak.com/wp-content/uploads/2015/12/JDK.png



Java Code Execution Process.

Java compiler ->

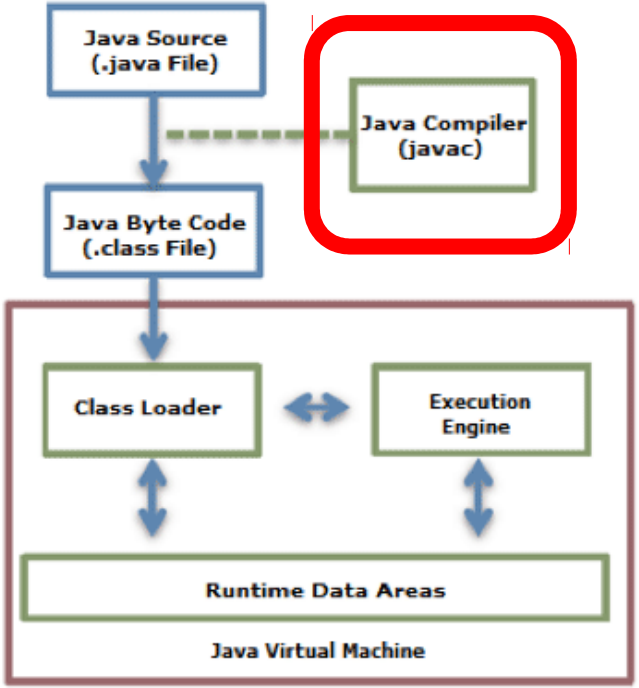
Compiler

Transform source code into bytecode

-
- .java -> .class

curiosity: Java compiler is written in… Java =)
runtime in ANSI C

info from: https://web.archive.org/web/20080222200518/http://java.sun.com/docs/overviews/java/java-overview-1.html



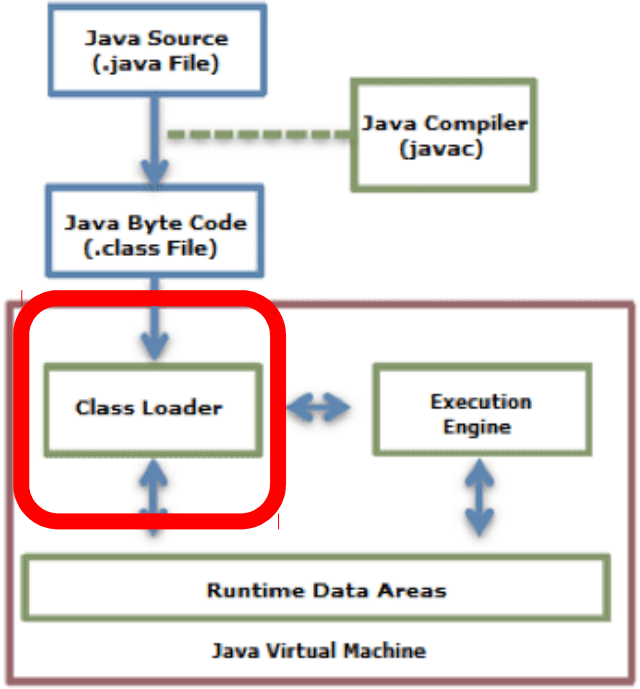
class loader ->

Class loader

A class loader loads the compiled Java Bytecode to the Runtime Data Areas.

Java provides a **dynamic load** feature; it loads and links the class **when it refers** to a class for the first time at runtime, not compile time. JVM's class loader executes the dynamic load.

There are many class loader rules, but let's stay with the basics.

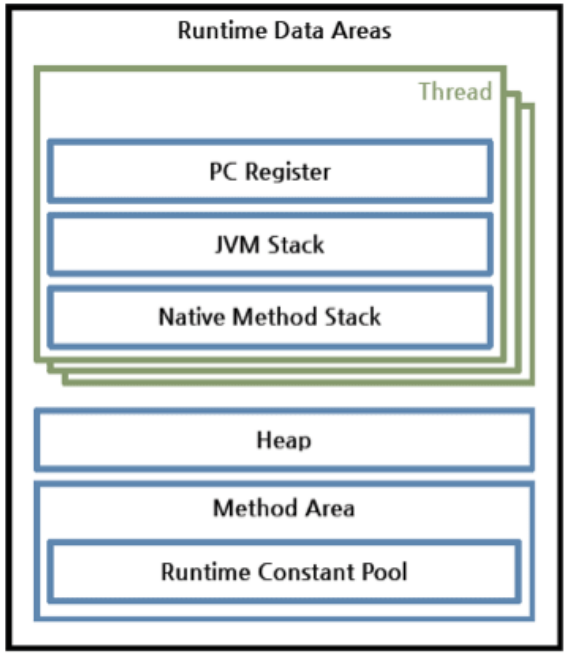


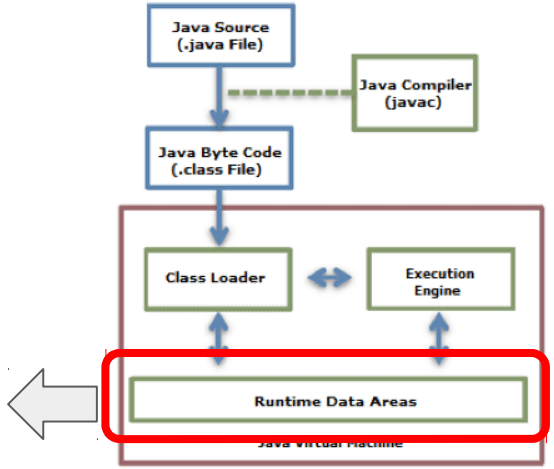
Runtime Data Areas ->

Runtime Data Areas

Runtime Data Areas are the memory areas assigned when the JVM program runs on the OS. The runtime data areas can be divided into 6 areas.

Per thread:
PC Register, JVM Stack, and Native Method Stack are created.
For all threads:
Heap, Method Area, and Runtime Constant Pool.





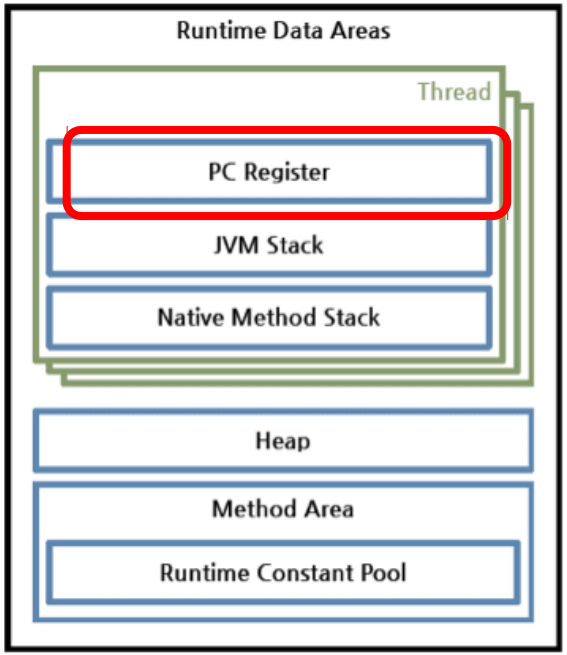
Each area shortly->

Each area shortly

PC register ->

PC register

- PC (Program Counter) has the address of a JVM instruction being executed now by the processor.

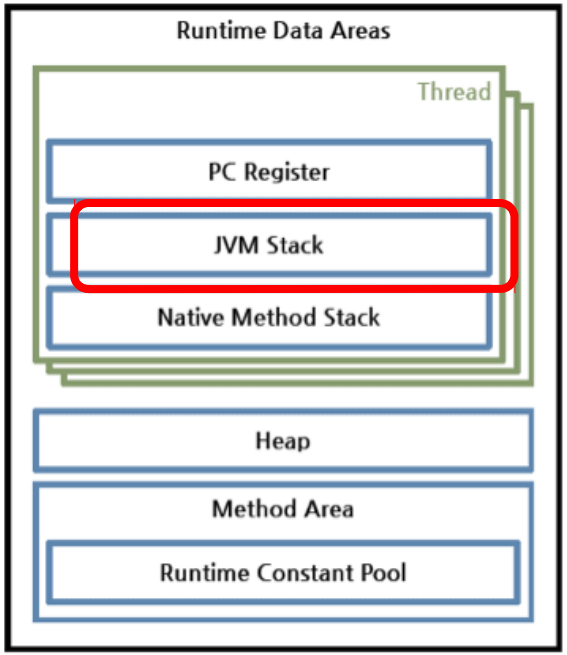


JVM Stack->

JVM Stack

- exists for each thread
- saves various stack frames
- The JVM just pushes or pops the stack frame to it.

If any exception occurs, each line of the stack trace shown as a method such as printStackTrace() expresses one stack frame.

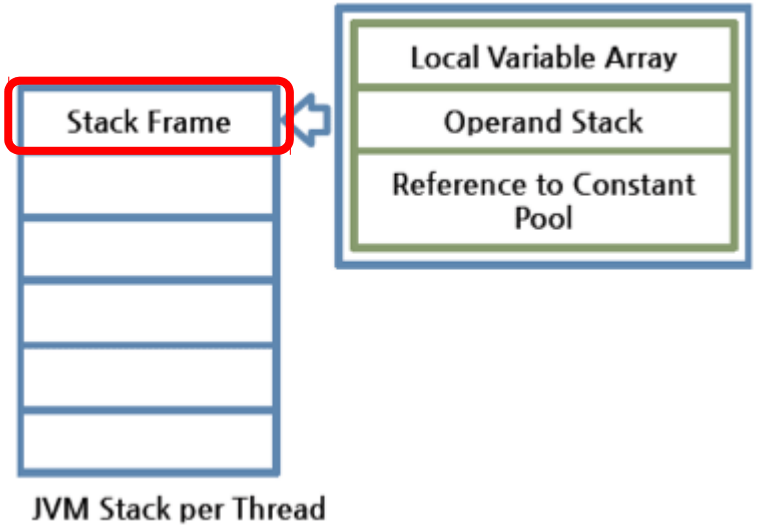


JVM Stack frame ->

Stack frame - veeery important!

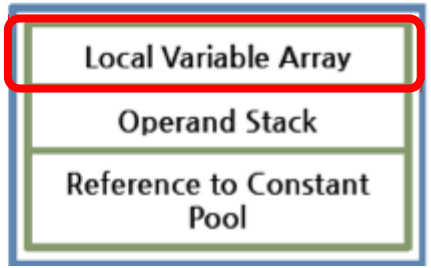
One stack frame is created whenever a method is executed in the JVM. When the method is ended, the stack frame is removed.

Each stack **frame** has the reference for local variable array, Operand stack, and runtime constant pool of a class where the method being executed belongs. The size of local variable array and Operand stack is determined while compiling, what you will see soon.



Local variable array in frame ->

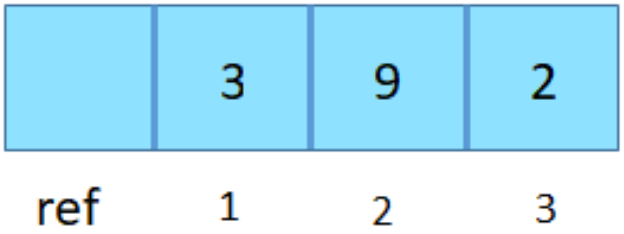
Local Variable Array



It has an index starting from 0. 0 - reference of a class instance where the method belongs.

From 1, the parameters sent to the method are saved.

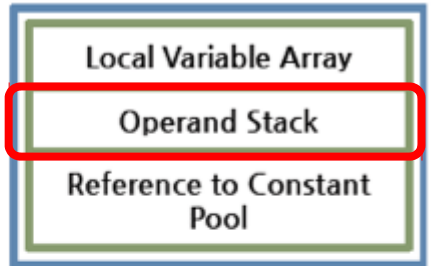
After the method parameters, the local variables of the method are saved.

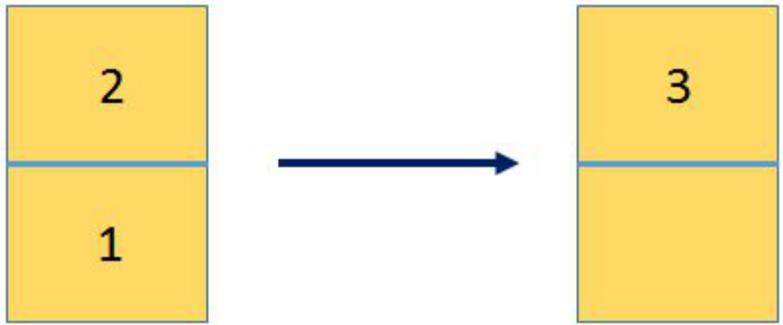


Operand Stack ->

Operand Stack

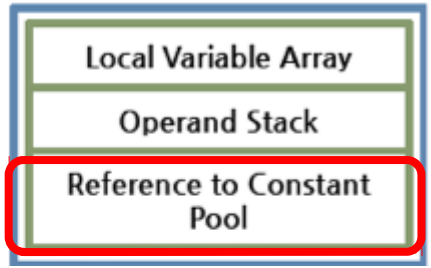
An actual workspace of a method. Each method exchanges data between the Operand stack and the local variable array, and pushes or pops other method invoke results. The necessary size of the Operand stack space can be determined during compiling. Therefore, the size of the Operand stack can also be determined during compiling.

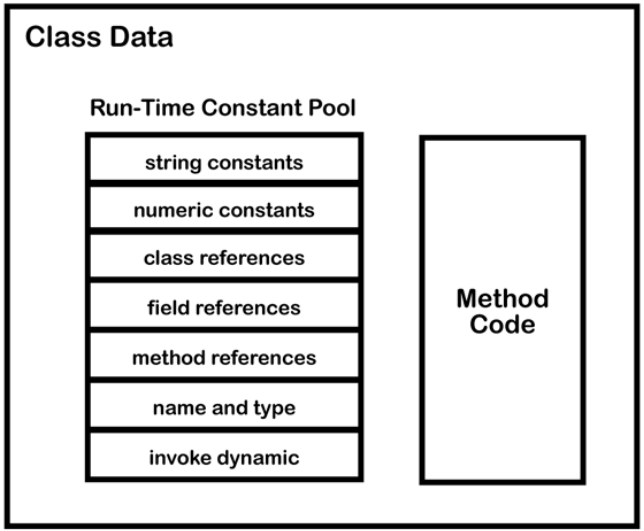




Reference to constant pool ->

Reference to Constant Pool





getfield #15

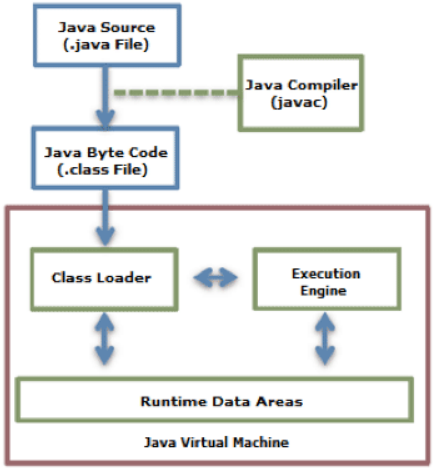
invokevirtual #23

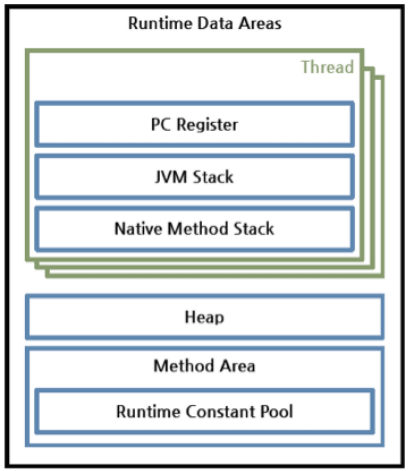
invokestatic #6

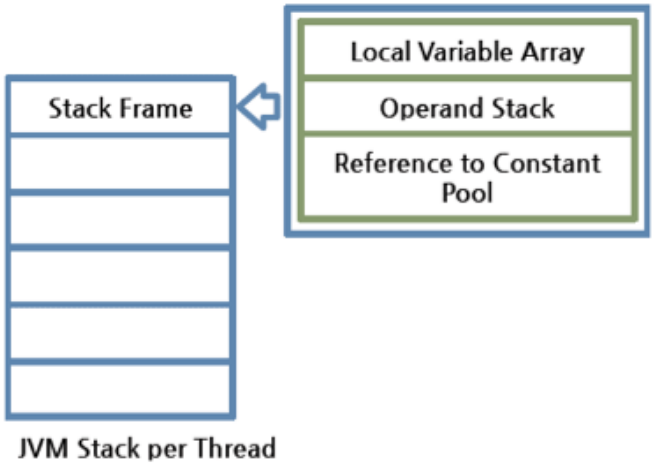
ldc #3

where are we with inception ->

Where are we right now with our inception?







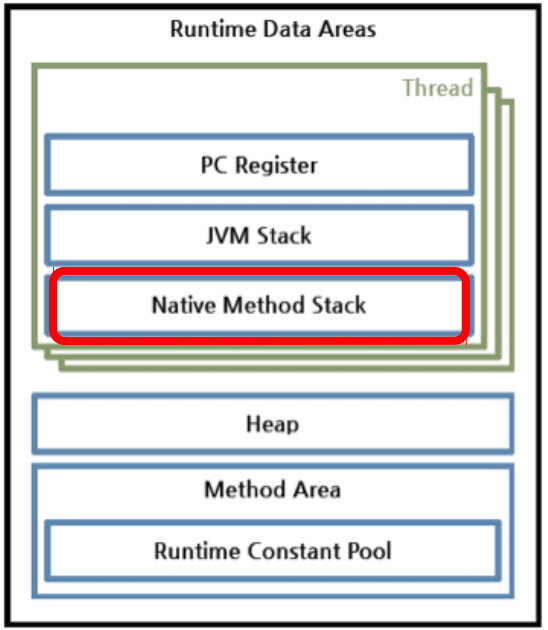
Native method stack ->

Native method stack

Native method stack: A stack for native code written in a language other than Java.

It is a stack used to execute C/C++ codes.

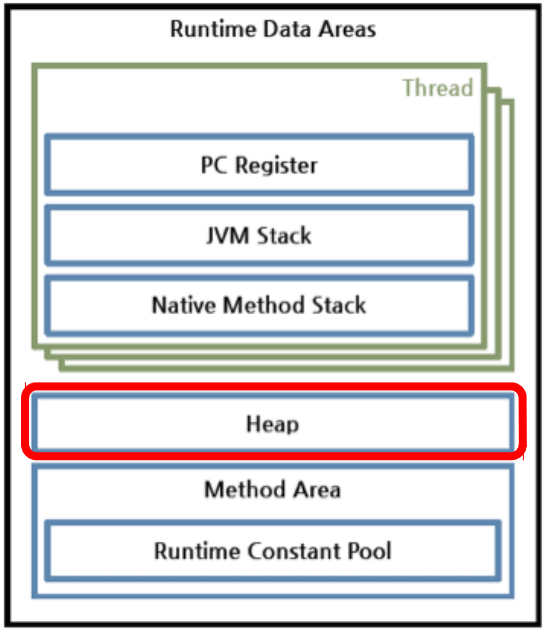
According to the language, a C stack or C++ stack is created.



Heap ->

Heap

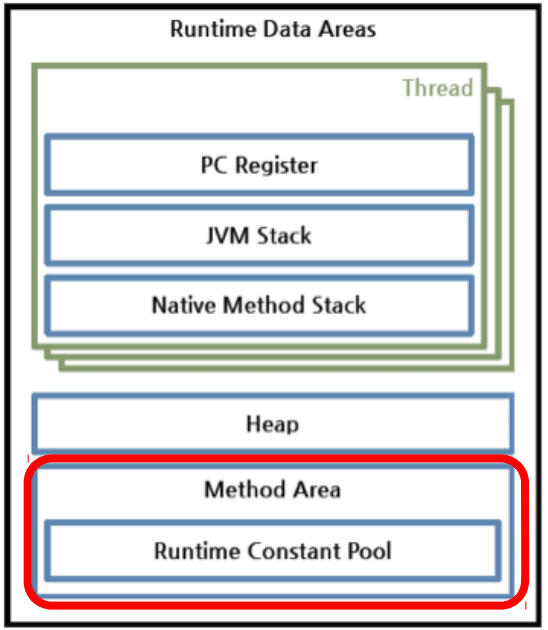
- Container for objects shared by all threads
- Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference.
- Any object created in the heap space has global access and can be referenced from anywhere of the application.



method area ->

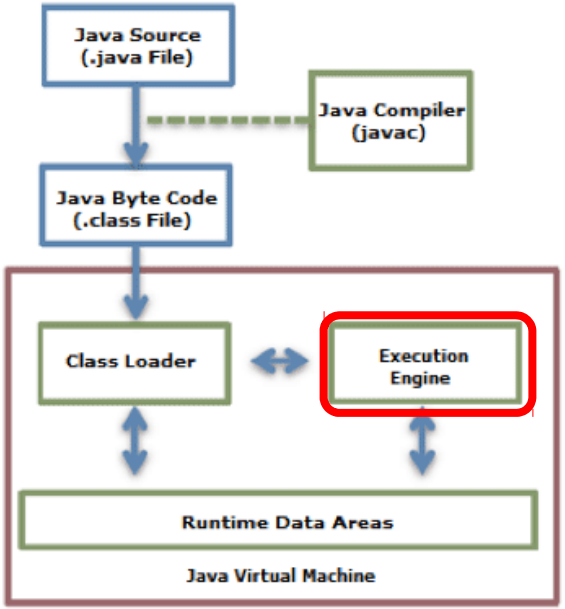
Method Area

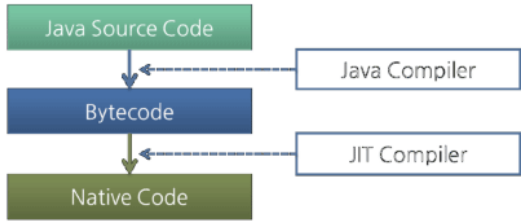
- Method area is shared by all threads, created when the JVM starts.
- It stores runtime constant pool, field and method information, static variable, and method bytecode for each of the classes and interfaces read by the JVM.



Execution engine

- Place where bytecode assigned to the runtime data areas is executed
- Every simple instruction is executed here one by one
- Bytecode is written in human language - execution engine transforms it into machine language
- Can be changed to machine code in two ways:
  - Interpreter
  - JIT (Just-In-Time) compiler





where are we with inception ->

Let’s practice

javap tool and ASM

javap:

- The **javap command** disassembles a class file.
- The javap command displays information about the fields,constructors and methods present in a class file.

ASM Bytecode outline - IntelliJ plugin

- Displays verbose bytecode for Java classes

Options of javap tool:

-c -prints out disassembled code

-p -shows all classes and members.

-v -(verbose)prints stack size, number of locals and args for methods.

-l -prints out line and local variable tables.

Exercise 1

Create empty class, compile it, and disassemble compiled file by javap -p. What can you notice?
You can find the solution on exercise1 branch.

Example 1: Test class

```
public class Test {
   public static void main(String[] args) {

      int a = 1;

      int b = 2;

      int c = a + b;

   }
}
```

Example 1: Test class

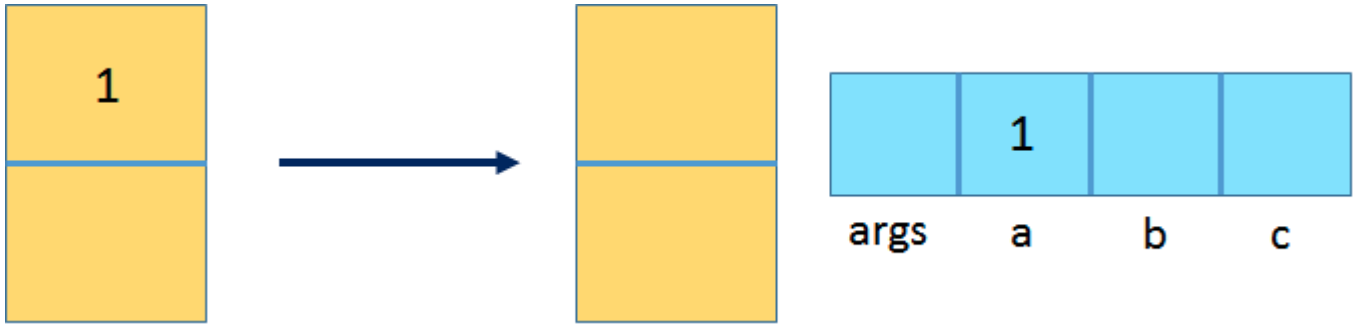
```
public class Test {
   public static void main(String[] args) {

      int a = 1;

      int b = 2;

      int c = a + b;

   }
}
```

Bytecode disassembled from Test.class

```
Compiled from "Test.java"
public class Test {
  public Test();
        Code:
        0: aload_0
        1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
        4: return

  public static void main(java.lang.String[]);
        Code:
        0: iconst_1
        1: istore_1
        2: iconst_2
        3: istore_2
        4: iload_1
        5: iload_2
        6: iadd
        7: istore_3
        8: return
}
```

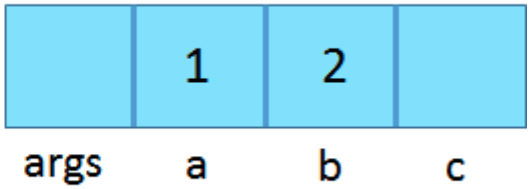
- iconst_1: Push the integer constant 1 onto the operand stack.

- istore_1: Pop the top operand (an int value) and store it in local variable at index 1, which corresponds to variable a.

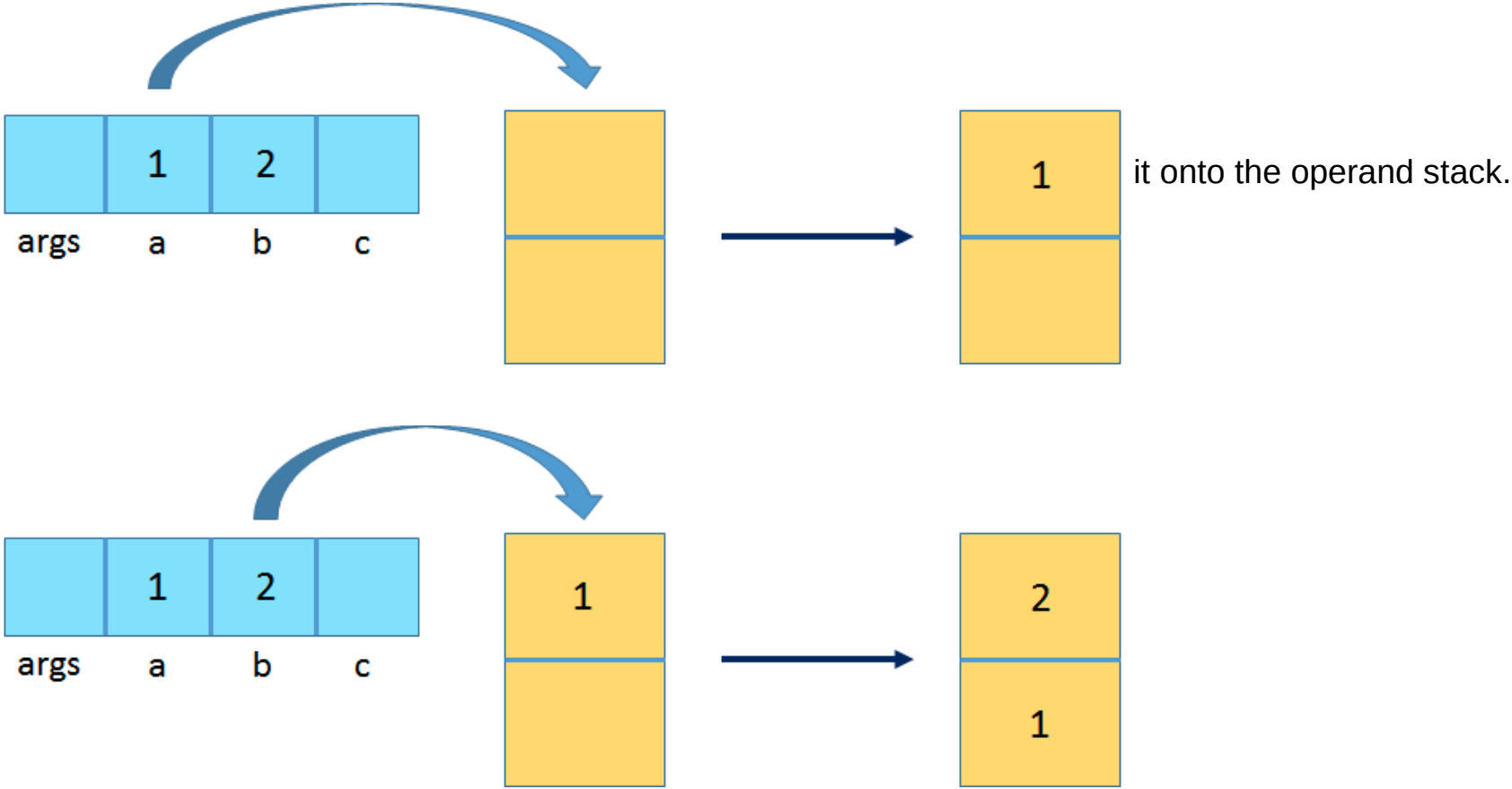


- iconst_2: Push the integer constant 2 onto the operand stack.

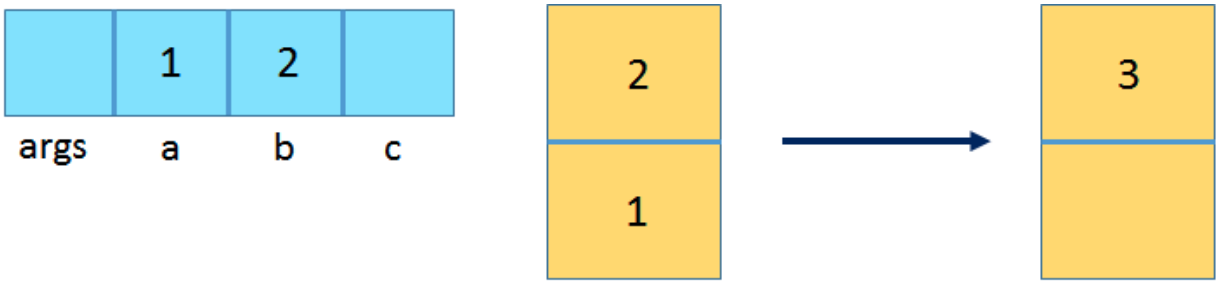
ue and store it in local variable at index 2, which corresponds



- iload_1: Load the int value from local variable at index 1 and push it onto the operand stack.



- iadd: Pop the top two int values from the operand stack, add them, and push the result back onto the operand stack.



Exercise 2

Write programme saying hello to user. As application parameter take name. Show generated bytecode. What interesting can you notice?

You can find the solution on exercise2 branch.

Exercise 3

Write below bytecode in Java:

```
public class exercises.Exercise3 {
  public exercises.Exercise3();
        Code:
        0: aload_0
        1: invokespecial #1                  // Method java/lang/Object."<init>":()V
        4: return

  public int someMethod(int);
        Code:
        0: bipush       22
        2: istore_2
        3: iload_1
        4: iload_2
        5: iadd
        6: ireturn
}
```

You can find the solution on exercise3 branch.

JVM languages

- used to produce computer software that runs on the Java virtual machine (JVM)
- code of those languages is compiled to Java bytecode
- Examples: Kotlin, Scala, JRuby, Jython

```
public class SampleClass {
  public SampleClass();
        Code:
        0: aload_0
        1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
        4: return


  public int met();
        Code:
        0: iconst_5
        1: istore_1
        2: bipush       7
        4: istore_2
        5: iload_1
        6: iload_2
        7: iadd
        8: istore_3
        9: iload_3
        10: ireturn
}
```

```
public class SampleClass {
  public SampleClass();
        Code:
        0: aload_0
        1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
        4: return


  public int met();
        Code:
        0: iconst_5
        1: istore_1
        2: bipush       7
        4: istore_2
        5: iload_1
        6: iload_2
        7: iadd
        8: istore_3
        9: iload_3
        10: ireturn
}
```

- Java

```
public class SampleClass {
        public int met() {
        int a = 5;
        int b = 7;
        int c = a + b;
        return c;
        }
}
```

- Kotlin
- class SampleClass {
- fun met(): Int {
- val a = 5
- val b = 7
- return a + b
- }

```
}
```

- Scala

```
class SampleClass {
  def met(): Int = {
        val a: Int = 5
        val b: Int = 7
        val c: Int = a + b
        c
  }
}
```

Example 2: Calculator class

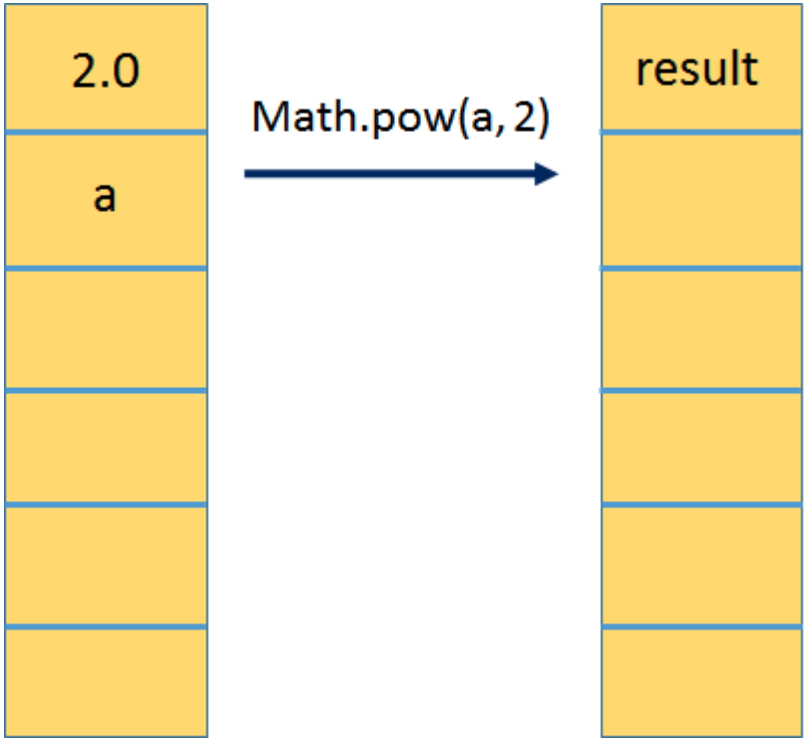
```java
public class Calculator {
  public static void main(String[] args) {

    int a = 1;

    int b = 2;

    int c = calc(a, b);

  }

  static int calc(int a, int b) {

    return (int) Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

  }
}
```

```
Compiled from "Calculator.java"
public class Calculator {
  public Calculator();
    Code:
      0: aload_0
      1: invokespecial #1                  // Method java/lang/Object."<init>":()V
      4: return

  public static void main(java.lang.String[]);
    Code:
      0: iconst_1
      1: istore_1
      2: iconst_2
      3: istore_2
      4: iload_1
      5: iload_2
      6: invokestatic  #2                  // Method calc:(II)I
      9: istore_3
     10: return

  static int calc(int, int);
    Code:
      0: iload_0
      1: i2d
      2: ldc2_w        #3                  // double 2.0d
      5: invokestatic  #5                  // Method java/lang/Math.pow:(DD)D
      8: iload_1
      9: i2d
     10: ldc2_w        #3                  // double 2.0d
     13: invokestatic  #5                  // Method java/lang/Math.pow:(DD)D
     16: dadd
     17: invokestatic  #6                  // Method java/lang/Math.sqrt:(D)D
     20: d2i
     21: ireturn
}
```

```
static int calc(int a, int b) {

    return (int) Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

  }
}
```

Math.pow(a, 2)

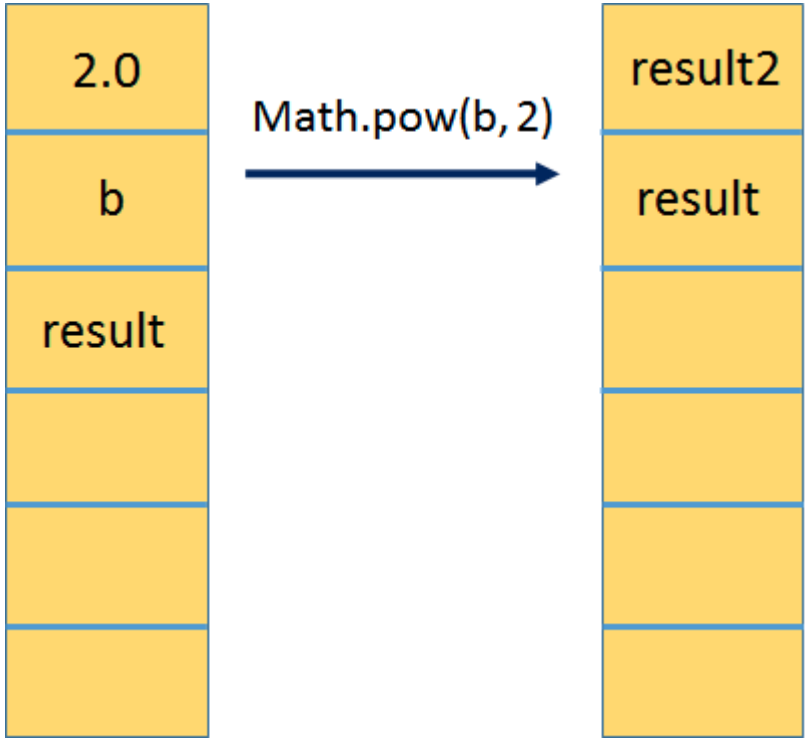
```
static int calc(int, int);
  Code:
    0: iload_0
    1: i2d
    2: ldc2_w        #3                  // double 2.0d
    5: invokestatic  #5                  // Method java/lang/Math.pow:(DD)D
    8: iload_1
    9: i2d
   10: ldc2_w        #3                  // double 2.0d
   13: invokestatic  #5                  // Method java/lang/Math.pow:(DD)D
   16: dadd
   17: invokestatic  #6                  // Method java/lang/Math.sqrt:(D)D
   20: d2i
   21: ireturn
```



```
static int calc(int a, int b) {

    return (int) Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

  }
}
```

Math.pow(b, 2)

```
static int calc(int, int);
  Code:
    0: iload_0
    1: i2d
    2: ldc2_w        #3                  // double 2.0d
    5: invokestatic  #5                  // Method java/lang/Math.pow:(DD)D
    8: iload_1
    9: i2d
   10: ldc2_w        #3                  // double 2.0d
   13: invokestatic  #5                  // Method java/lang/Math.pow:(DD)D
   16: dadd
   17: invokestatic  #6                  // Method java/lang/Math.sqrt:(D)D
   20: d2i
   21: ireturn
```



Conditions

if-else condition

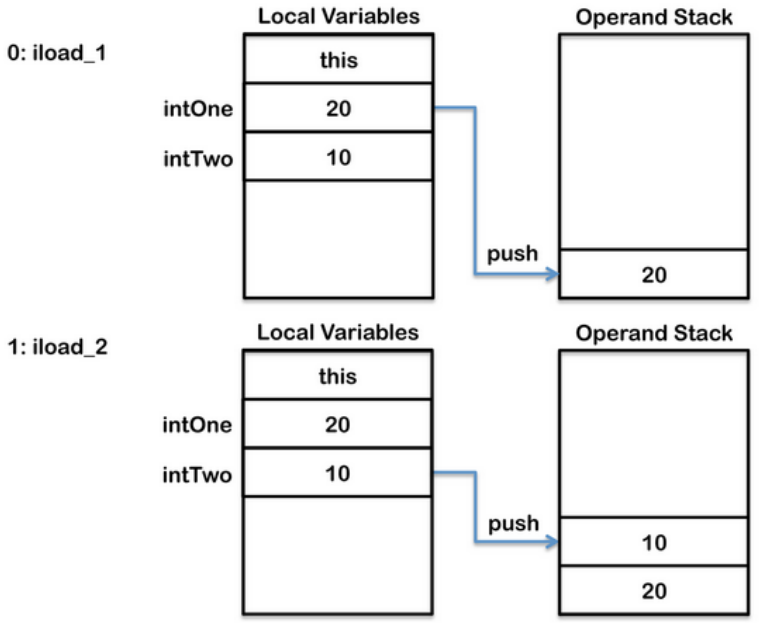
```
public class IfElse {
   public int greaterThen(int intOne, int intTwo) {
       if (intOne > intTwo) {
           return 0;
       } else {
           return 1;
       }
   }
}
```

```
Compiled from "IfElse.java"
public class IfElse {
  public IfElse();
        Code:
        0: aload_0
        1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
        4: return


  public int greaterThen(int, int);
        Code:
        0: iload_1
        1: iload_2
        2: if_icmple    7
        5: iconst_0
        6: ireturn
        7: iconst_1
        8: ireturn
}
```

```java
public class IfElse {
  public int greaterThen(int intOne, int intTwo) {
    if (intOne > intTwo) {
      return 0;
    } else {
      return 1;
    }
  }
}
```

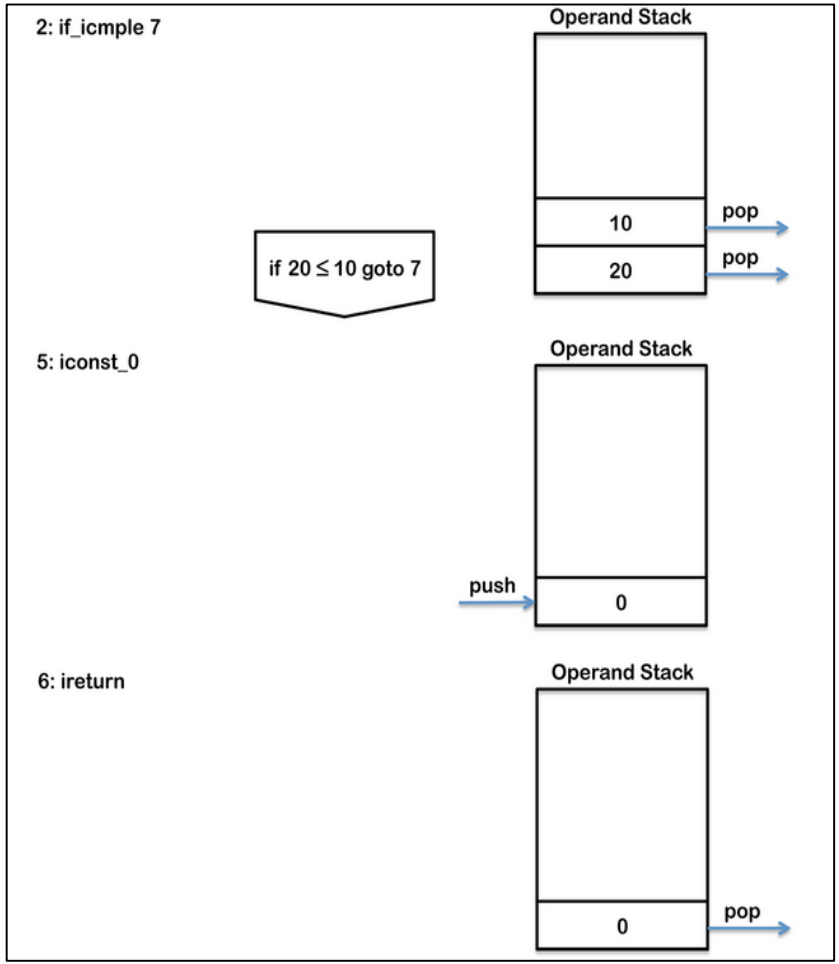
```
public int greaterThen(int, int);
    Code:
    0: iload_1
    1: iload_2
    2: if_icmple        7
    5: iconst_0
    6: ireturn
    7: iconst_1
    8: ireturn
}
```

```
public int greaterThen(int intOne, int intTwo) {
    if (intOne > intTwo) {
      return 0;
    } else {
      return 1;
    }
}

greaterThen(10, 20);
```



```java
public class IfElse {
   public int greaterThen(int intOne, int intTwo) {
      if (intOne > intTwo) {
         return 0;
      } else {
         return 1;
      }
   }
}
```

```
  public int greaterThen(int, int);
        Code:
        0: iload_1
        1: iload_2
        2: if_icmple        7
        5: iconst_0
        6: ireturn
        7: iconst_1
        8: ireturn
}
```



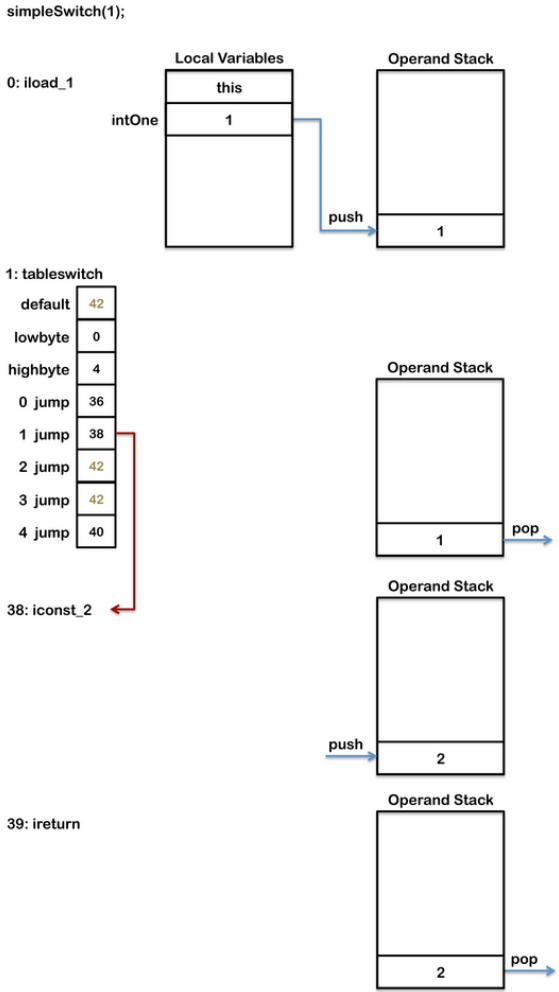
Switch condition

```
public class Switch {
   public int simpleSwitch(int intOne) {
      switch (intOne) {
         case 0:
            return 3;
         case 1:
            return 2;
         case 4:
            return 1;
         default:
            return -1;
      }
   }
}
```

```
Compiled from "Switch.java"
public class conditions.Switch {
  public conditions.Switch();
            Code:
            0: aload_0
            1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
            4: return


  public int simpleSwitch(int);
            Code:
            0: iload_1
            1: tableswitch   { // 0 to 4
                      0: 36
                      1: 38
                      2: 42
                      3: 42
                      4: 40
            default: 42
            }
            36: iconst_3
            37: ireturn
            38: iconst_2
            39: ireturn
            40: iconst_1
            41: ireturn
            42: iconst_m1
            43: ireturn
}
```

```java
public class Switch {
  public int simpleSwitch(int intOne) {
    switch (intOne) {
      case 0:
        return 3;
      case 1:
        return 2;
      case 4:
        return 1;
      default:
        return -1;
    }
  }
}
```

```
public int simpleSwitch(int);
        Code:
        0: iload_1
        1: tableswitch   { // 0 to 4
                0: 36
                1: 38
                2: 42
                3: 42
                4: 40
        default: 42
        }
        36: iconst_3
        37: ireturn
        38: iconst_2
        39: ireturn
        40: iconst_1
        41: ireturn
        42: iconst_m1
        43: ireturn
}
```



Loops

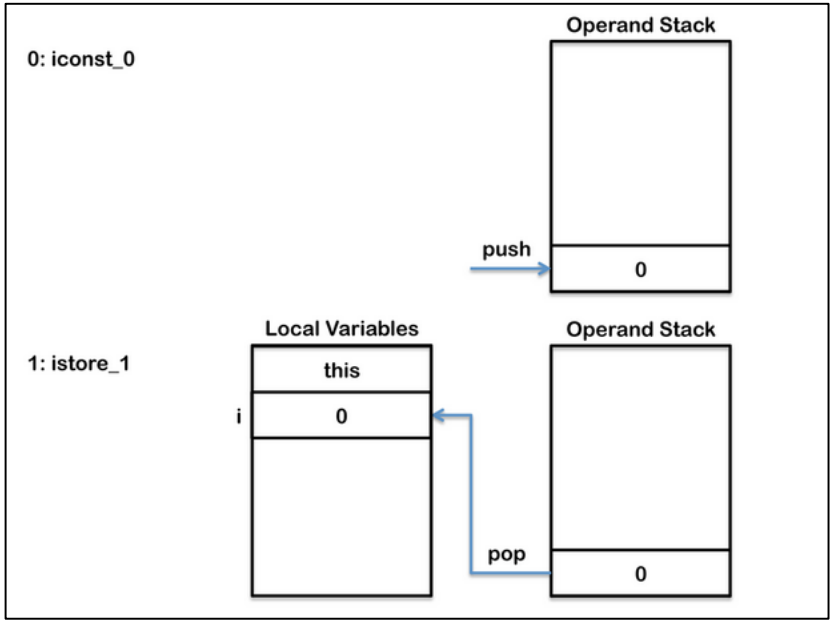
While loop

```
public class While {


   public void whileLoop() {
       int i = 0;
       while (i < 2) {
           i++;
       }
   }


}
```

```
Compiled from "While.java"
public class loops.While {
  public loops.While();
            Code:
            0: aload_0
            1: invokespecial #1                  // Method java/lang/Object."<init>":()V
            4: return


  public void whileLoop();
            Code:
            0: iconst_0
            1: istore_1
            2: iload_1
            3: iconst_2
            4: if_icmpge       13
            7: iinc            1, 1
            10: goto           2
            13: return
}
```

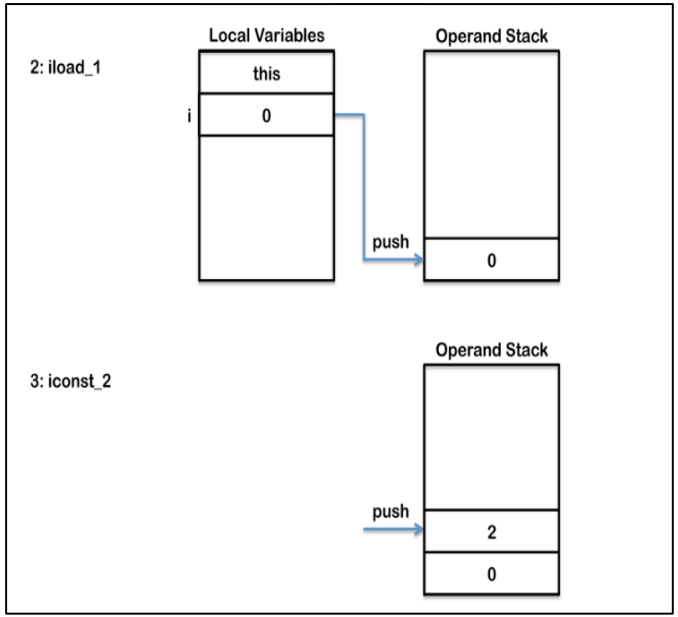
```java
public class While {


   public void whileLoop() {
       int i = 0;
       while (i < 2) {
           i++;
       }
   }


}
```

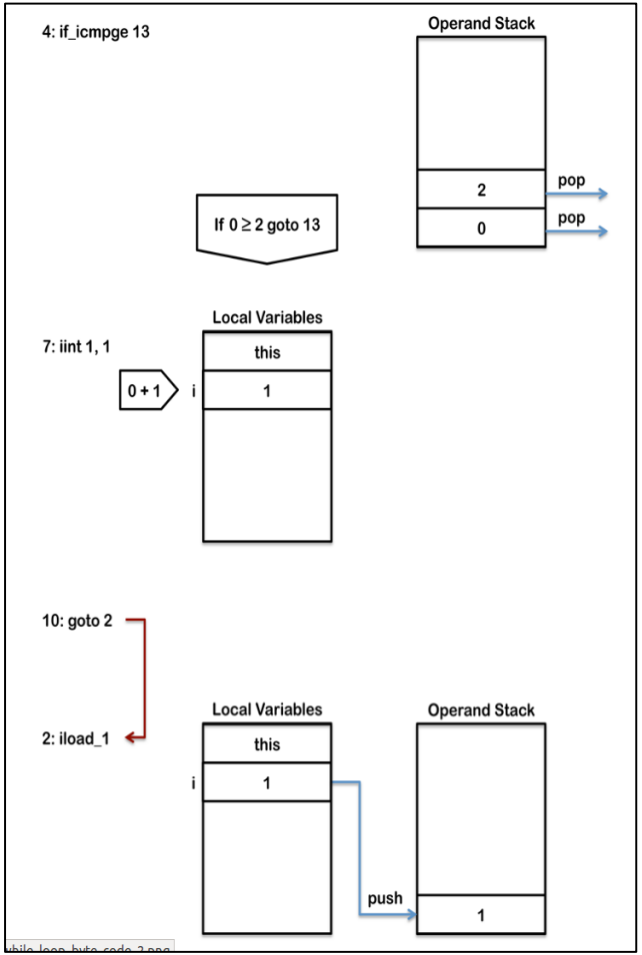
```
public void whileLoop();
        Code:
        0: iconst_0
        1: istore_1
        2: iload_1
        3: iconst_2
        4: if_icmpge  13
        7: iinc       1, 1
        10: goto      2
        13: return
```



```
public class While {


   public void whileLoop() {
      int i = 0;
      while (i < 2) {
         i++;
      }
   }


}
```

```
public void whileLoop();
        Code:
        0: iconst_0
        1: istore_1
        2: iload_1
        3: iconst_2
        4: if_icmpge  13
        7: iinc       1, 1
        10: goto      2
        13: return
```





For loop

```
public class ForLoop {
   static void forLoop() {
      for (int i = 0; i < 5; i++) {
      }
   }
}
```

```
Compiled from "ForLoop.java"
public class loops.ForLoop {
  public loops.ForLoop();
        Code:
        0: aload_0
        1: invokespecial #1                  // Method java/lang/Object."<init>":()V
        4: return


  static void forLoop();
        Code:
        0: iconst_0
        1: istore_0
        2: iload_0
        3: iconst_5
        4: if_icmpge     13
        7: iinc          0, 1
        10: goto         2
        13: return
}
```

Exercise 4 Do-While

Rewrite and Compile below class. Explain how generated bytecode works.

```
public class Exercise4 {

   public void doWhileLoop() {
      int i = 0;
      do {
         i++;
      } while (i < 2);
   }
}
```

You can find the solution on exercise4 branch.

Do-While loop

```
public class Exercise4 {

   public void doWhileLoop() {
      int i = 0;
      do {
         i++;
      } while (i < 2);
   }
}
```

```
Compiled from "DoWhile.java"
public class loops.DoWhile {
  public loops.DoWhile();
        Code:
        0: aload_0
        1: invokespecial #1
// Method java/lang/Object."<init>":
()V
        4: return

  public void doWhileLoop();
        Code:
        0: iconst_0
        1: istore_1
        2: iinc          1, 1
        5: iload_1
        6: iconst_2
        7: if_icmplt 2
        10: return
}
```

Exercise 5

Anonymous vs lambda

Compile following class. See the difference between opcode generated by implementation of anonymous class and lambda.

You can find the solution on exercise5 branch.

```
public class Exercise4 {

  void anonymousImpl() {

    Thread t1 = new Thread(new Runnable() {
      @Override
      public void run() {

      }
    });
  }


  void lambdaImpl() {
    Thread t2 = new Thread(() -> {

    });
  }
}
```

Try-catch-finally

```
public class TryCatchFinally {
   public void tryCatchCatchFinally(int i) {
      try {
         i = 2;
      } catch (RuntimeException e) {
         i = 3;
      } finally {
         i = 4;
      }
   }
}
```

```
public class objectOriented.TryCatchFinally {
  public objectOriented.TryCatchFinally();
            Code:
            0: aload_0
            1: invokespecial #1                  // Method java/lang/Object."<init>":()V
            4: return


  public void tryCatchCatchFinally(int);
            Code:
// try block
            0: iconst_2
            1: istore_1
// finally block - no exception
            2: iconst_4
            3: istore_1
            4: goto          20
// catch block
            7: astore_2
            8: iconst_3
            9: istore_1
// finally block - after catch block
            10: iconst_4
            11: istore_1
            12: goto         20
// finally block - after exception not caught in catch block
            15: astore_3
            16: iconst_4
            17: istore_1
            18: aload_3
            19: athrow
            20: return
            Exception table:
            from    to  target type
// jump to catch block for RuntimeException
            0       2       7   Class java/lang/RuntimeException
  // jump to finally block if any exception (other than RuntimeException)
            0       2       15   any
  // jump to finally block if exception in catch block
            7       10      15   any
}
```

Synchronized block vs synchronized method

```
public class SynchronizedBlock {


   public void synchronizedBlock(int i) {
      synchronized (this) {
         i = 1;
      }
   }
}
```

```
public class SynchronizedMethod {


   public synchronized void synchronizedMethod(int i) {
      i = 1;
   }
}
```

Please look at objectOriented branch.

```
public class objectOriented.SynchronizedBlock {
  public objectOriented.SynchronizedBlock();
            Code:
            0: aload_0
            1: invokespecial #1                  // Method java/lang/Object."<init>":()V
            4: return


  public void synchronizedBlock(int);
            Code:
            0: aload_0
            1: dup
            2: astore_2
            3: monitorenter
            4: iconst_1
            5: istore_1
            6: aload_2
            7: monitorexit
            8: goto          16
            11: astore_3
            12: aload_2
            13: monitorexit
            14: aload_3
            15: athrow
            16: return
            Exception table:
            from    to  target type
            4       8       11   any
            11      14      11   any
}
```

```
Compiled from "SynchronizedMethod.java"
public class objectOriented.SynchronizedMethod {
  public objectOriented.SynchronizedMethod();
            Code:
            0: aload_0
            1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
            4: return


  public synchronized void synchronizedMethod(int);
            Code:
            0: iconst_1
            1: istore_1
            2: return
}
```

Project Lombok

1. Maven dependency

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.16.20</version>
    <scope>provided</scope>
</dependency>
```

2. Intellij Lombok plugin

Please look at lombok branch.

Lombok and Vanilla Java

```java
@AllArgsConstructor
@ToString
public class PersonWithLombok {
   private String firstName;
}
```

```java
public class PersonVanillaJava {
   private String firstName;


   public PersonVanillaJava(String firstName) {
      this.firstName = firstName;
   }


   @Override
   public String toString() {
      return "PersonVanillaJava(firstName=" + this.firstName + ")";
   }
}
```

Vanilla Java:

```
Compiled from "PersonVanillaJava.java"
public class lombok.PersonVanillaJava {
  public lombok.PersonVanillaJava(java.lang.String);
        Code:
        0: aload_0
        1: invokespecial #1        // Method java/lang/Object."<init>":()V
        4: aload_0
        5: aload_1
        6: putfield      #2        // Field firstName:Ljava/lang/String;
        9: return


  public java.lang.String toString();
        Code:
        0: new           #3        // class java/lang/StringBuilder
        3: dup
        4: invokespecial #4        // Method java/lang/StringBuilder."<init>":()V
        7: ldc       #5            // String PersonVanillaJava(firstName=
        9: invokevirtual #6        // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
        12: aload_0
        13: getfield     #2        // Field firstName:Ljava/lang/String;
        16: invokevirtual #6       // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
        19: ldc          #7        // String )
        21: invokevirtual #6       // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
        24: invokevirtual #8       // Method java/lang/StringBuilder.toString:
()Ljava/lang/String;
        27: areturn
}
```

Lombok:

```
Compiled from "PersonWithLombok.java"
public class lombok.PersonWithLombok {
  public lombok.PersonWithLombok(java.lang.String);
        Code:
        0: aload_0
        1: invokespecial #1        // Method java/lang/Object."<init>":()V
        4: aload_0
        5: aload_1
        6: putfield      #2        // Field firstName:Ljava/lang/String;
        9: return


  public java.lang.String toString();
        Code:
        0: new           #3        // class java/lang/StringBuilder
        3: dup
        4: invokespecial #4        // Method java/lang/StringBuilder."<init>":()V
        7: ldc       #5            // String PersonWithLombok(firstName=
        9: invokevirtual #6        // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
        12: aload_0
        13: getfield     #2        // Field firstName:Ljava/lang/String;
        16: invokevirtual #6       // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
        19: ldc          #7        // String )
        21: invokevirtual #6       // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
        24: invokevirtual #8       // Method java/lang/StringBuilder.toString:
()Ljava/lang/String;
        27: areturn
}
```

Disable annotation processor

add to pom.xml:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>8</source>
        <target>8</target>
      </configuration>
      <executions>
        <execution>
          <id>default-compile</id>
          <configuration>
            <compilerArgument>-proc:none</compilerArgument>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

After disabling annotation processor

Lombok:

```
Compiled from "PersonWithLombok.java"
public class lombok.PersonWithLombok {
  public lombok.PersonWithLombok();
    Code:
    0: aload_0
    1: invokespecial #1                  // Method java/lang/Object."<init>":()V
    4: return
}
```

Vanilla Java:

```
Compiled from "PersonWithLombok.java"
public class lombok.PersonWithLombok {
  public lombok.PersonWithLombok(java.lang.String);
    Code:
    0: aload_0
    1: invokespecial #1                  // Method java/lang/Object."<init>":()V
    4: aload_0
    5: aload_1
    6: putfield      #2                  // Field firstName:Ljava/lang/String;
    9: return


  public java.lang.String toString();
    Code:
    0: new           #3                  // class java/lang/StringBuilder
    3: dup
    4: invokespecial #4                  // Method java/lang/StringBuilder."<init>":()V
    7: ldc           #5                  // String PersonWithLombok(firstName=
    9: invokevirtual #6                  // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
    12: aload_0
    13: getfield     #2                  // Field firstName:Ljava/lang/String;
    16: invokevirtual #6                 // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
    19: ldc          #7                  // String )
    21: invokevirtual #6                 // Method java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
    24: invokevirtual #8                 // Method java/lang/StringBuilder.toString:
()Ljava/lang/String;
    27: areturn
}
```

Refs:

- https://www.cubrid.org/blog/understanding-jvm-internals/
- https://dzone.com/articles/introduction-to-java-bytecode

Authors

@epam
Krzysztof Dzioba

@epam
Krzysztof Sroczyk