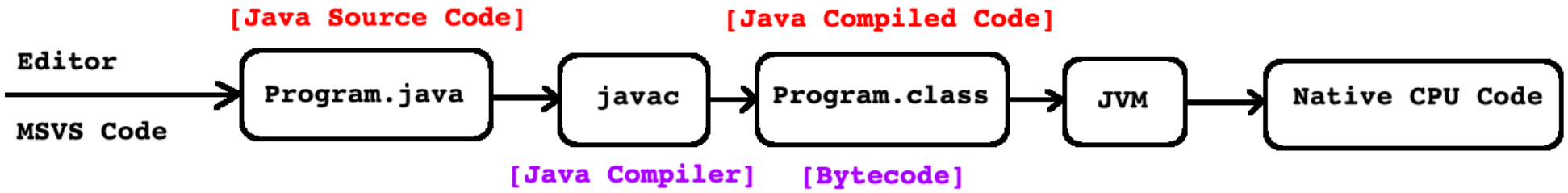
Object Oriented Programming with Java 8
PG-DAC

Rohan Paramane

Agenda

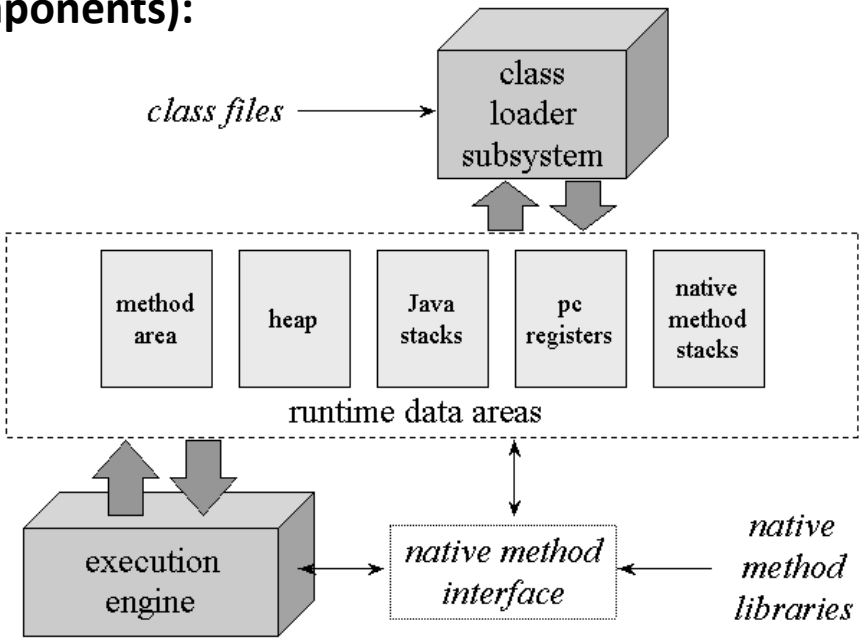
- JVM Architecture
- Command Line Compilation
- STS Hello World
- Datatypes
  - Primitive and Non Primitive
- Control Flow statements
  - if
  - else
  - for
  - while
  - do..while

Java application flow of execution



Components of JVM (Major 3 Components):

1. **Class Loader Sub System**
   - Bootstrap class loader
   - Extension classLoader
   - Application classLoader
   - User Defined class Loader
2. **Runtime Data Areas**
3. **Execution Engine**
   - Interpreter
   - Compiler
   - Garbage Collector



[JVM Architecture]

Method Area

- Metaspace: JDK 1.8 onwards Loaded Byte codes (Loaded class info) 1 Single copy static data members, constructors, methods

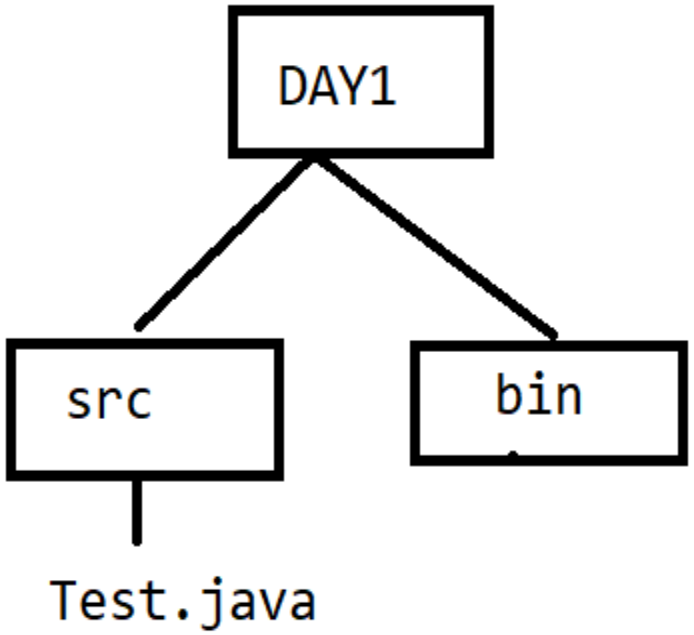
Heap

- Java object heap state of the object (non static data members) 1 single copy

Java Stack

- Stack is created one per thread , method local info(like args,local vars, ret vals) Individual stack : stack frames

How to compile the code from src and store .class inside bin




```
D:\DAY1> cd src
D:\DAY1\src>javac –d ..\bin Test.java
D:\DAY1\src>cd ..\bin
D:\DAY1\bin>java Test
```

Java Keywords

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
true	false	null			

Data Types

- Data type of any variable decide following things:
  1. **Memory:** How much memory is required to store the data.
  2. **Nature:** Which kind of data is allowed to store inside memory.
  3. **Operation:** Which operations are allowed to perform on the data stored in memory.
  4. **Range:** Set of values that we can store inside memory.

- The Java programming language is a statically typed language, which means that every variable and every expression has a type that is known at compile time.
  - Types of data type:
    1. **Primitive type(also called as value type)**
       - **boolean** type
       - Numeric type
         1. Integral types(**byte, char, short, int, long**)
         2. Floating point types(**float, double**
    2. **Non primitive type(also called as reference type)**
       - **Interface, Class, Type variable, Array**

Variables

- A **variable** is a name given memory location. That memory is associated to a data type and can be assigned a value.
- int n; float f1; char ch; double d;

Rules of Variables

- All variable names must begin with a letter of the alphabet, an underscore (_), or a dollar sign ($). Can't begin with a digit. The rest of the characters may be any of those previously mentioned plus the digits 0-9.

- The convention is to always use a (lower case) letter of the alphabet. The dollar sign and the underscore are discouraged.

```
1. int n1;
2. n1 =21 ;          // assignment
3. int val=50; //initialization
5. double d = 21.8;   // initialization
6. d = n1;           // assignment
7. float f1 = 16.13F;
```

Data Types

Sr.No.	Primitive Type	Size[In Bytes]	Default Value[For Field]	Wrapper Class
1	boolean	Isn't specified	FALSE	Boolean
2	byte	1	0	Byte
3	char	2	\u0000	Character
4	short	2	0	Short
5	int	4	0	Integer
6	float	4	0.0f	Float
7	double	8	0.0d	Double
8	long	8	0L	Long

Note : Char datatype supports UNICODE character set, so 2 bytes .

Operators

- Arithmetic Operators
- Unary Operators
- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators
- Shift Operators

Operators Cont..

- **Arithmetic Operators**
  - They are used to perform simple arithmetic operations on primitive data types.

  * : Multiplication

  / : Division

  % : Modulo

  + : Addition

  – : Subtraction
- **Unary Operators**
  - Unary operators need only one operand. They are used to increment, decrement or negate a value.
  - Unary minus, used for negating the values.
  - eg : int a=20; int b=-a;
  - ++ :Increment operator, used for incrementing the value by 1. There are two varieties of increment operator.
  - Post-Increment : Value is first used for computing the result and then incremented.
  - Pre-Increment : Value is incremented first and then result is computed.
  - -- : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operator.
  - Post-decrement : Value is first used for computing the result and then decremented.
  - Pre-Decrement : Value is decremented first and then result is computed.
  - ! : Logical not operator, used for inverting a boolean value.
  - eg : boolean jobDone=true; boolean flag=!jobDone; System.out.println(flag);

Operators Cont..

- **Assignment Operators**
  - ‘=’ Assignment operator is used to assign a value to any variable. It has a right to left associativity.
  - Eg. int val = 500;
  - assignment operator can be combined with other operators to build a shorter version of statement called Compound Statement.
  - +=, for adding left operand with right operand and then assigning it to variable on the left.
  - -=, for subtracting left operand with right operand and then assigning it to variable on the left.
  - *=, for multiplying left operand with right operand and then assigning it to variable on the left.
  - /=, for dividing left operand with right operand and then assigning it to variable on the left.
  - %=, for assigning modulo of left operand with right operand and then assigning it to variable on the left.
- **Relational Operators**
  - These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are used in looping statements and conditional if else statements.
  - ==, Equal to : returns true if left hand side is equal to right hand side.
  - !=, Not Equal to : returns true if left hand side is not equal to right hand side.
  - <, less than : returns true if left hand side is less than right hand side.
  - <=, less than or equal to : returns true if left hand side is less than or equal to right hand side.
  - >, Greater than : returns true if left hand side is greater than right hand side.
  - >=, Greater than or equal to: returns true if left hand side is greater than or equal to right hand side.

Operator Cont…

- **Logical Operators :**
  - These operators are used to perform “logical AND” and “logical OR” operation, i.e. the function similar to AND gate and OR gate in digital electronics.
    &&, Logical AND : returns true when both conditions are true.
    ||, Logical OR : returns true if at least one condition is true.
  - eg : int data=100;
    int data2=50;
    if(data > 60 && data2 < 100)
    System.out.println("test performed...");
    else
    System.out.println("test not performed...");

- **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary.
  - General format is :
    condition ? if true : if false
    The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.
    eg : int data=100;
    System.out.println(data>100?"Yes":"No");

Operators Cont..

- **Bitwise Operators :**
  - These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.
  - &, Bitwise AND operator: returns bit by bit AND of input values.
  - |, Bitwise OR operator: returns bit by bit OR of input values.
  - ^, Bitwise XOR operator: returns bit by bit XOR of input values.
  - ~, Bitwise Complement Operator: This is a unary operator which returns the one's compliment representation of the input value, i.e. with all bits inversed.
  - Eg : String binary[] = { "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
    "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111" };

```
int a = 3; // 0 + 2 + 1 or 0011 in binary
int b = 6; // 4 + 2 + 0 or 0110 in binary
int c = a | b;
int d = a & b;
int e = a ^ b;
System.out.println("      a = " + binary[a]);
System.out.println("      b = " + binary[b]);
System.out.println("    a|b = " + binary[c]);
System.out.println("    a&b = " + binary[d]);
System.out.println("    a^b = " + binary[e]);      }
```

Operators Cont…

- **Shift Operators :**
  - These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two.

  - <<, Left shift operator: shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.

eg :

```
int a = 25;
System.out.println(a<<4); //25 * 16 = 400
a=-25;
System.out.println(a<<4);//-25 * 16 = -400
```

  - Signed right shift operator : The signed right shift operator '>>' uses the sign bit to fill the trailing positions. For example, if the number is positive then 0 will be used to fill the trailing positions and if the number is negative then 1 will be used to fill the trailing positions.

Example Shift Operations

- Assume if a = 60 and b = -60; now in binary format, they will be as follows −

- a = 0000 0000 0000 0000 0000 0000 0011 1100
- b = 1111 1111 1111 1111 1111 1111 1100 0100
- In Java, negative numbers are stored as 2's complement.

- Thus a >> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
- And b >> 1 = 1111 1111 1111 1111 1111 1111 1110 0010

- Unsigned right shift operator
- The unsigned right shift operator '>>' do not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.

- Thus a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
- And b >>> 1 = 0111 1111 1111 1111 1111 1111 1110 0010

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements: Conditional Stmt: if, if else, switch

Looping: for, while, do while

Escapes: break, continue, return

if Statement – different syntax options

```
if
(expression)
   statement;
```

⇨ A single statement.

```
if
(expression)
   {
    statements;
   }
```

⇨ A block of statements.

```
if
(expression)
   statement;
else
statement;
```

⇨ Single statement in the if and a single statement in the else.

```
if
(expression)
      statement;
else
{
   statements;
}
```

⇨ A single statement in the if and a block of statements in the else.

Conditional Operator

- The operator “ ? : ” is the only operator that takes three operands, each of which is an expression.
- The value of the whole expression equals the value of expr2 if expr1 is true, or equals the value of expr3 if expr1 is false.
- Syntax:

```
expr1 ? expr2 : expr3
```

switch Statement

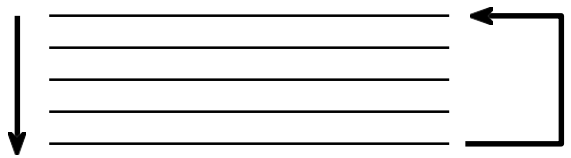
- The nested if can become complicated and unreadable.
- The switch statement is an alternative to the nested if.
- Syntax:

```
switch(expression)
{
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    case constant expr:
            statement(s);
            [break;]
    default :
            statement(s);
            [break;]
}
```

- Usually, but not always, the last statement of a case is break.
- default case is optional.

Loops

- Loops break the serial execution of the program.
- A group of statements is executed a number of times.



There are three kinds of loops :

- **`while`**
- **`for`**
- **`do … while`**

While – Loop

- Syntax:

```
while (expression)
   Statement;
```

or

```
while (expression)
{Statements;}
```

- The loop continues to iterate as long as the value of expression is true (expression differs from zero).
- Expression is evaluated each time before the loop body is executed.
- The braces { } are used to group declarations and statements together into a compound statement or block, so they are syntactically equivalent to a single statement.

for - Loop

- Syntax:

```
for (expr1 ; expr2 ; expr3)
     statement;
```

or

```
for (expr1 ; expr2 ; expr3)
{
    statements;
}
```

- Is equivalent to:

```
expr1;
while (expr2)
{
    {statements;}
    expr3;
}
```

do while Loop

- Syntax:

```
do
{
    Statements;
}while (expression);
```

- The condition expression for looping is evaluated only after the loop body had executed.

break Statement

- We have seen how to use the break statement within the switch statement.

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

continue Statement

- In some situations, you might want to skip to the next iteration of a loop without finishing the current iteration.
- The **`continue`** statement allows you to do that.
- When encountered, **`continue`** skips over the remaining statements of the loop, but **`continues`** to the next iteration of the loop.

Thank you.

Rohan.paramane@sunbeaminfo.com