JAVA

UNIT - 1

CLASS NOTES

feedback/corrections: vibha@pesu.pes.edu

Vibha Masti

JAVA

- platform, language

features

- used to create **applets** (small applications)
- does **not require preprocessors**
- **multi-threaded** language (uninterrupting)
  eg: websites: download from server as well as showing GUI
  Process: any executing program
  Thread: unit of a process
- **object oriented** (class, objects, inheritance etc.)
- **platform-independent** (OS, microprocessor features make a platform), unlike C/C++

prog.java → javac → byte code → JVM → output

.java extension

compiler

.class intermediate code ← one for each class

(platform independent)

interpreter

- **simple language** (unlike C — pointer handling) there is no concept of pointers (JVM will handle), no operator overloading, strong in-built garbage collector, in-built exception handling.
- **robust** language

TERMINOLOGY

Java Development Toolkit (JDK)

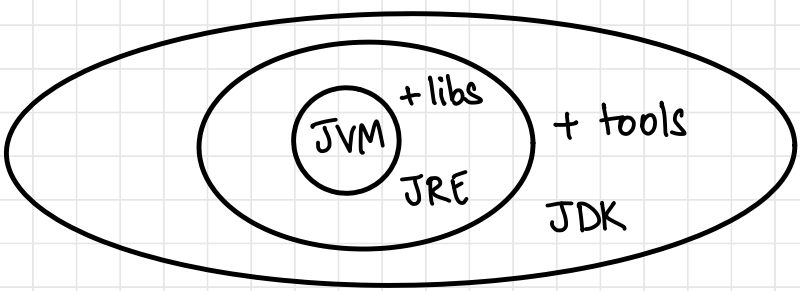
- to run/develop java programs

Java Runtime Environment (JRE)

- not for development; mainly for execution

Java Virtual Machine (JVM)

- interpreter that makes it platform independent
- execution engine
- makes it platform independent (WORA) ← write once read anywhere



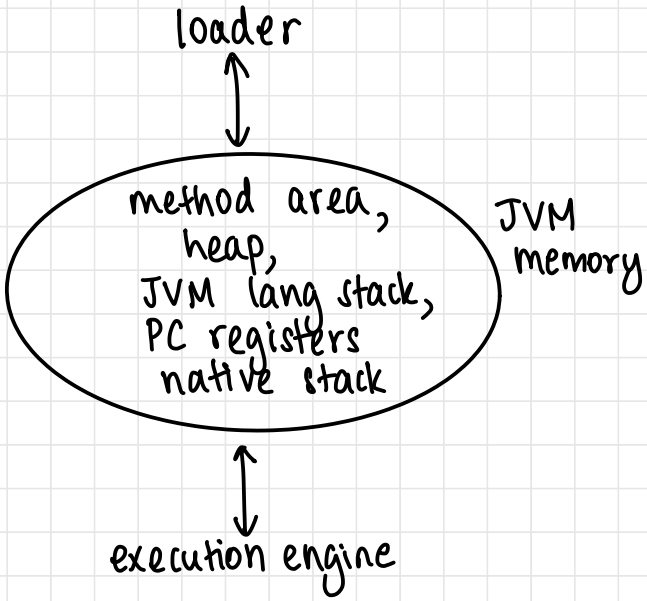
COMPILER vs INTERPRETER

- analysis takes longer on compiler, but execution is quicker

Syllabus

1+2 : intro, terminology (OOP), recap of control structs
3+4 : arrays, strings, inheritance, compositions, package
5 : nested types, generics, files

JVM - 3 PARTS



(A) Loader

1. Loading

- .class file converted to binary file
- class instances created and stored in heap

2. Linking

- linking
- verification — from valid compiler
- initialising default variables

3. Initialisation

- values in code to variables will be initialised

(B) JVM Memory

1. Method area
- info about class (name, name of parent, methods, static variables)
- Only one method for entire program (shared)

2. Heap
- dynamically allocated memory
- object / references created
- shared among threads

3. Stack
- threads running in background
- block of stack ← activation record (info on method call - return address, arguments)
- one thread ⟶ one stack
- at the end, thread destroyed

4. Registers
- Program counter (PC) register
- shared among threads

(C) Execution Engine

1. Interpreter
- analyses line by line and executes o/p
- problem: when same function called multiple times, JIT helps

2. Just In Time Compiler (JIT)
   - compiles code and gives binary data
   - when same function called multiple times

3. Garbage collection
   - handled by JVM; programmer does not have to allocate/free memory
   - happen in the background

VARIABLES

- container to hold values in memory
- identifier to access — rules (can start with $ and _; can't start with digit, can't be a keyword)
- case-sensitive language

DATATYPES

- size (bytes), type of data, range of values
- operations to perform
- string - " " (combination of characters & digits)
- char - ' '
- boolean - true & false

```
data_type  var_name (id) = value;
```

Access Specifiers for Variables (Note: pkg – folder)

	private	default	protected	public
declaration	private int a;	int a;	protected int a;	public int a;
same class	yes	yes	yes	yes
same pkg, sub class	no	yes	yes	yes
same pkg, non sub class	no	yes	yes	yes
diff pkg, sub class	no	no	yes	yes
diff pkg, non sub class	no	no	no	yes

DIFFERENT CLASSES OF DATATYPES

1) Primitive
2) Non-primitive

PRIMITIVE

Data types	Size (bytes)
byte	1
short	2
int	4
long	8
float	4
double	8
char	2
boolean	1 bit

- Size is fixed
- Pre-defined in JVM by developers
- Passed by value
- No pre-defined functions
- Starts with lowercase (convention)
- Pre-defined

NON-PRIMITIVE

- Pre-defined functions available
- User-defined
- Must be associated with a value (null also)
- starts with uppercase (convention)
- passed by reference in function arguments
- also called reference variables
- same size — Object
- string is pre-defined; all others must be user-defined

type casting

1) Widening
   - happens automatically by JVM
   - assigning smaller value to larger datatype
   - eg: assigning byte value to short value

2) Narrowing
   - done manually
   - assigning larger value to smaller data
   - eg: assigning double to float value

Mixed Mode Operations

- arithmatic operations (+, -, *, /, ==)
- float & double → widening occurs
- arithmatic operations (+, -, *, /, %)
- byte, short, int, char → widening occurs

- mixed mode: float & long (+, -, *, /)
  long → float (8 bit to 4 bit)

Operators

1) Arithmatic
   - ↳ +, -, *, /, %, ++, --
   - ↳ =
   - ↳ +=, -=, ...
   - ↳ <, <=, >, >=, ==, !=

2) Logical Operators
   - ↳ &&, ||, !
   - ↳ &, |, ~, ^, <<, >>, >>> ← zero filled shift right (bitwise)

Control Statements

```
↳ if (cond) {...}
↳ else {...}
↳ else if (cond) {...}

↳ switch (expression) {
        case x: break;
          ⋮
  }
```

Loops

```
↳ while (condition) {...}
↳ for ( st1; st2; st3) {...}
↳ for ( type var: array {...}
```

(init; condition; update)
for each; like C++ range-based for
for (auto i: v) {...}

eg:
```
int a[] = {1, 2, 3};

for (int i : a) {
    System.out.print(i + " ");
}
```

Loop Keywords

- break
- continue

Command Line Arguments

```
String args[]
```

Abstraction

- objects

Encapsulation

- data + methods wrapped

Inheritance

- acquire properties from parent

Classes

non-primitive data type

```
class class-name {
    int length, breadth, height;
}
```

instance variables