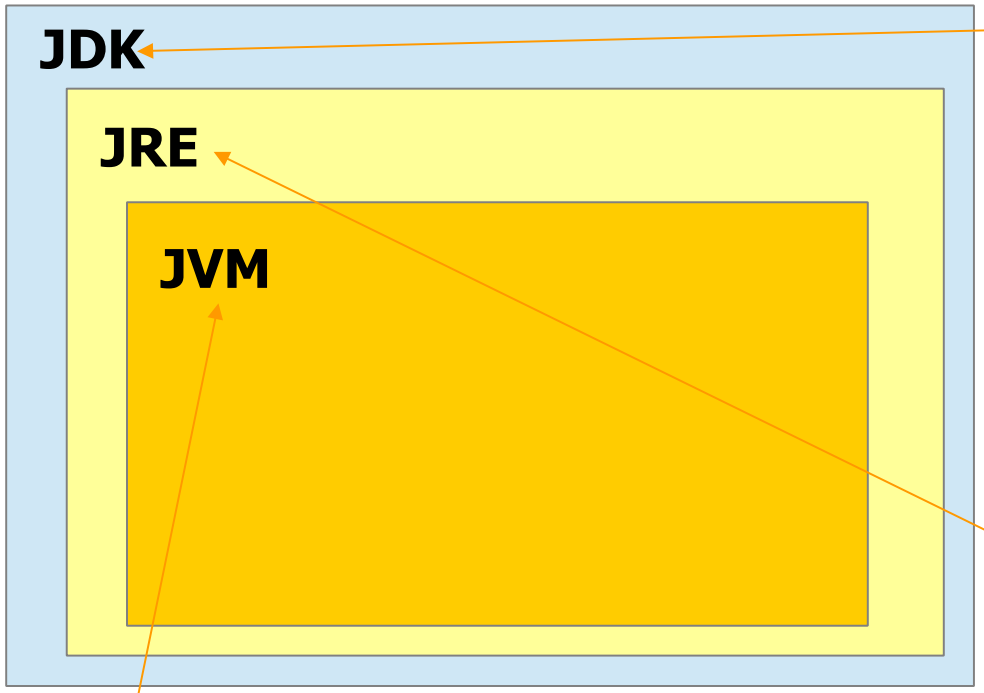
ARCHITETTURA JAVA
(MODULO JAVA BASE)

JVM, JRE, JDK



JDK: Java Development Kit, è una estensione della JRE che contiene questa, ma è inoltre un insieme di software che potete utilizzare per sviluppare le applicazioni basate su Java. La Java Development Kit è necessaria per sviluppare nuove applicazioni Java.

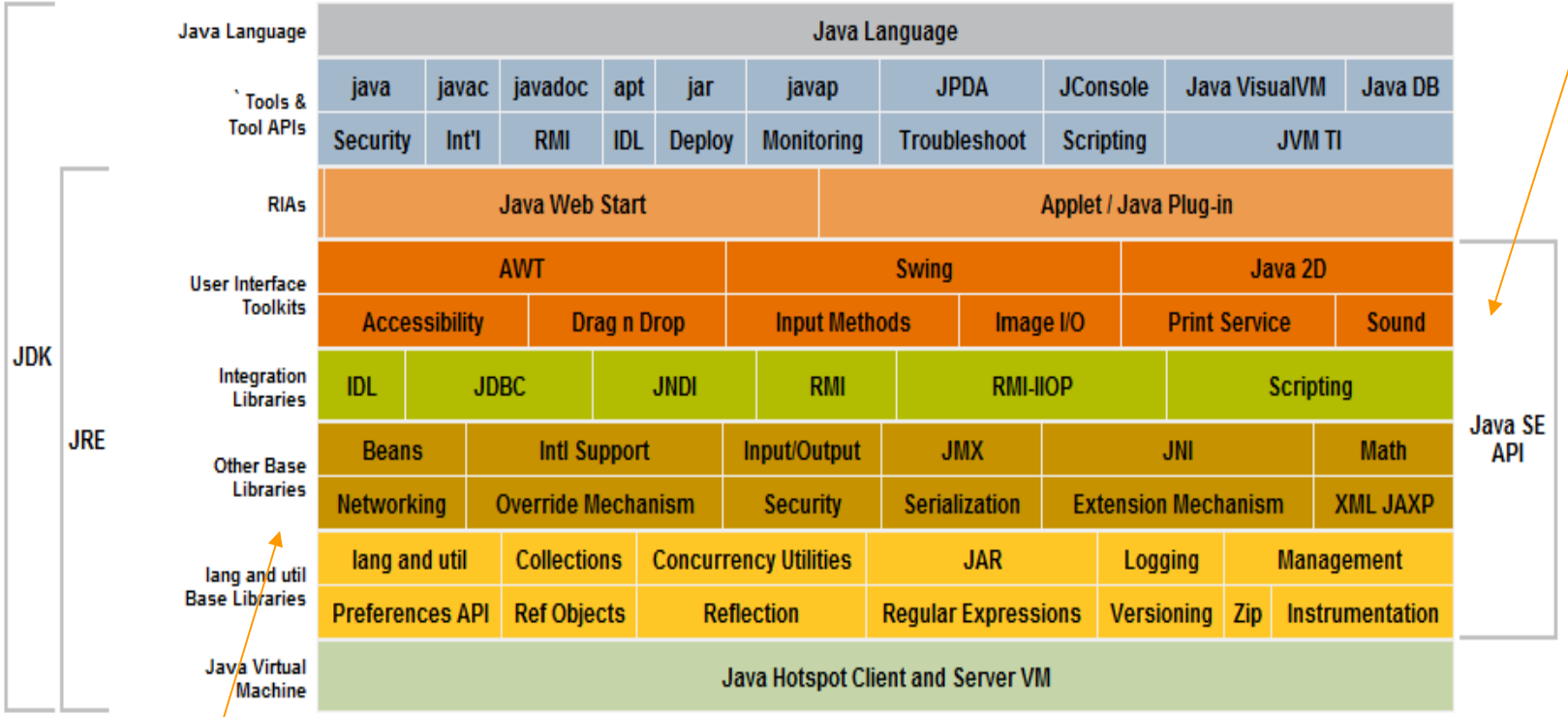
La differenza principale con la JRE è che contiene un compilatore Java per trasformare i nostri codici java in bytecode, cioè in file di tipo .class. E' possibile richiamare il compilatore Java tramite il comando javac da linea di comando.

La JDK richiede più spazio sul disco poiché contiene JRE ed inoltre l'insieme delle classi API, il compilatore Java, Web Start e i file aggiuntivi necessari per scrivere le applet e le applicazioni Java.

JRE: Java Runtime Environment, è un'implementazione della Macchina virtuale Java, specifica per un S.O. ed un'architettura hardware, che esegue effettivamente i programmi Java. Include oltre la JVM, delle librerie di base e altri componenti aggiuntivi per eseguire le applicazioni e le applet scritte in Java. In poche parole ci permette di eseguire una qualsiasi applicazione Java.

JVM: Java Virtual Machine, è il componente della piattaforma Java che esegue i programmi tradotti in fase di compilazione in bytecode, il pseudo-codice interpretabile dalla JVM. JVM ha il compito di interpretare il bytecode eseguendone le istruzioni. Per ragioni di efficienza, in alternativa la JVM può tradurre (sul momento) in codice macchina il bytecode, mediante un compilatore JIT (Just In Time)

Java Platform SE



Java SE API + JVM=> Java Platform Standard Edition, ampiamente utilizzata nella programmazione in linguaggio Java per costruire e distribuire applicazioni portatili (di tipo Desktop Application) ad uso generale. Java SE consiste di una macchina virtuale (Java Virtual Machine), che deve essere usata per eseguire programmi Java, insieme ad una serie di librerie (o "pacchetti") o API necessari per consentire l'uso di file system, reti, interfacce grafiche e così via, all'interno di tali programmi.

JAVA SE API: le librerie standard mirano ad includere le necessità più comuni per il programmatore. Fra le più significative si possono citare

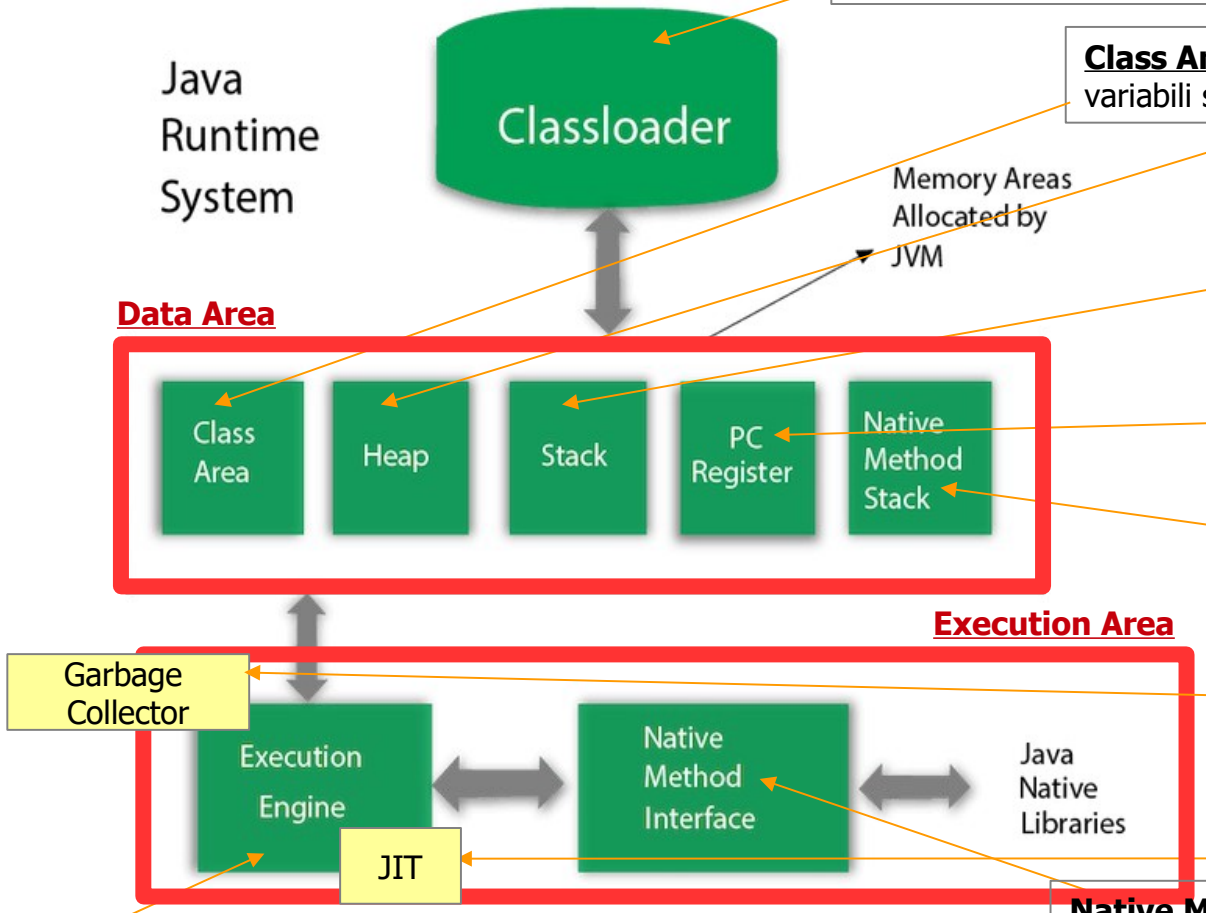
- la possibilità di costruire GUI (interfacce grafiche) con strumenti standard e non proprietari, utilizzando package quali Java Swing e JavaFX;
- la possibilità di creare applicazioni multi-threaded, ovvero che svolgono in modo concorrente molteplici attività;
- il supporto per la riflessione, ovvero la capacità di un programma di agire sulla propria struttura e di utilizzare classi caricate dinamicamente dall'esterno.

La vastità delle librerie standard contribuiscono a rendere l'uso di Java altamente integrabile con altre tecnologie. Alcuni esempi delle funzionalità incluse tra le librerie standard di Java sono:

- accesso ai database tramite JDBC;
- manipolazione documenti XML;
- dialogo con piattaforme CORBA;
- potenti strumenti per la programmazione lato server nel contesto Web;
- supporto nativo per gran parte dei protocolli della famiglia IP, vedi ad esempio il Socket Java;
- supporto per le applicazioni multimediali, streaming audio e video.

La **Data Area** dalla JVM `e concettualmente divisa in tre parti: **Class Area**: area di memoria in cui vengono caricate (allocate) tutte le classi che costituiscono il programma; **Stack**: area di memoria in cui vengono caricati (allocati) i record di attivazione dei metodi, e quindi tutte le variabili locali; **Heap**: Area di memoria in cui vengono caricati (allocati) tutti i vari oggetti creati nel programma.

Java Virtual Machine



Classloader: carica il file .class, precentemente compilato con javac, nella JVM.

Class Area: una delle Data Area della JVM, nella quale sono memorizzati i dati della classe come variabili statiche, blocchi di istruzioni statici, metodi ed i riferimenti a loro (istanze).

Heap: allocazione della memoria di runtime per gli oggetti. Contiene lo stato dell'oggetto istanziato, definito dalle variabili membro.

Stack: area di memoria in cui vengono caricati (allocati) i record di attivazione dei metodi (riferimenti ai metodi) così come le variabili locali

PC Register: registro contatore contenente l'indirizzo di un'istruzione che è attualmente in esecuzione nella JVM.

Native Method Stack: analogo a **Stack** ma usato nelle chiamate delle Java Native Libraries (mediante Native Method Interface).

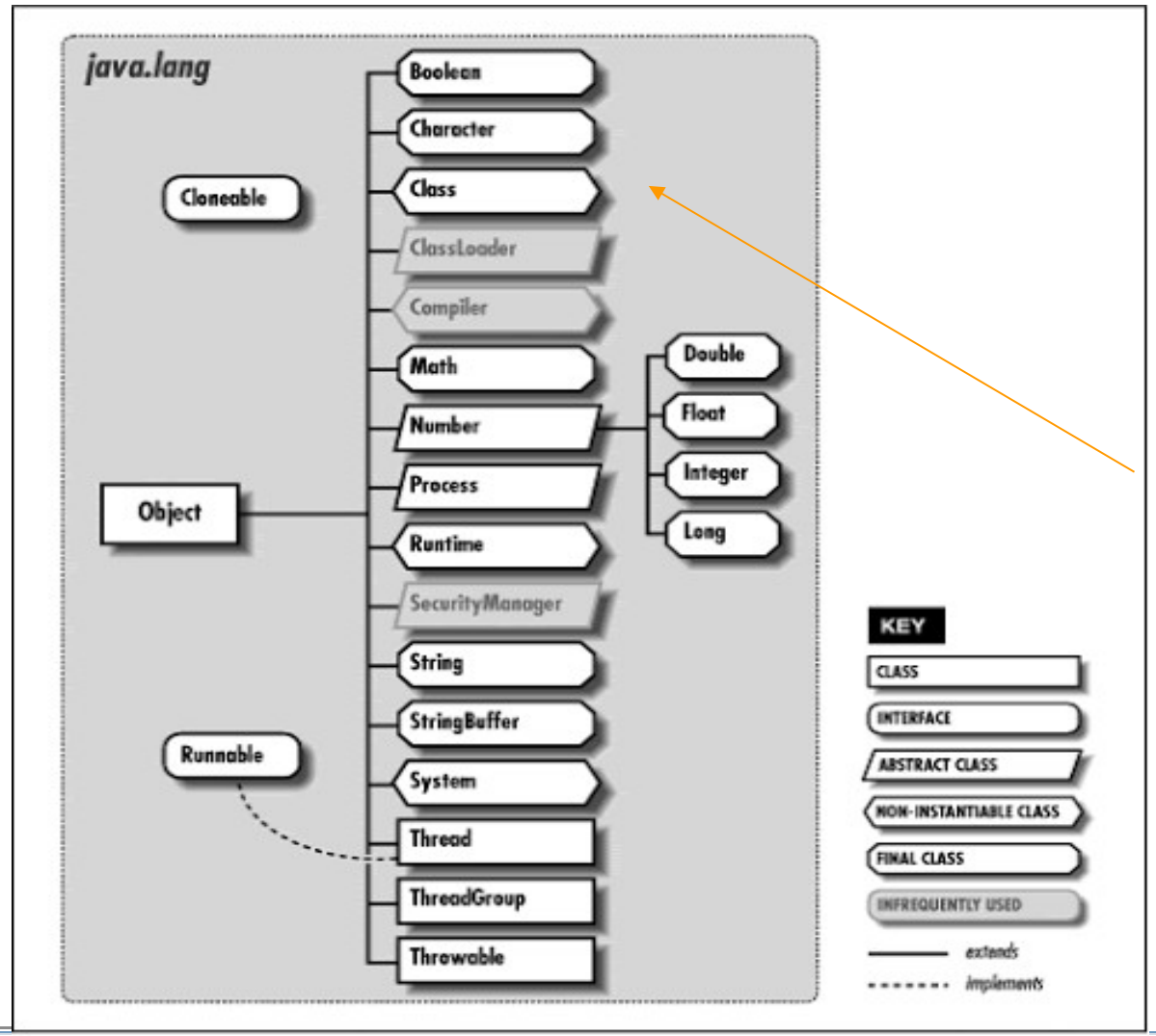
Garbage Collector: modalità automatica di gestione della memoria, mediante la quale la JVM a run-time libera porzioni di memoria non più utilizzate dalle applicazioni. In particolare il Garbage Collector rimuove i riferimenti agli oggetti non più utilizzati

JIT: compilatore Just In Time, traduce istruzioni da byte-code a codice macchina.

Native Method Inteface: interfaccia verso la Java Native Libraries che consente al codice Java di richiamare codice cosiddetto "nativo", ovvero specifico di un determinato sistema operativo su cui la JVM è in esecuzione.

Execution engine: contiene un processore virtuale, per interpretare le istruzioni bytecode una ad una e un compilatore JIT (Just In Time).

Java SE API, package principali



Java possiede un'enorme libreria di classi standard(API) che costituisce uno dei punti di forza del linguaggio. Tale libreria è organizzata in un sistema di vari packages (pacchetti) che raccolgono le classi secondo un'organizzazione basata sul campo d'utilizzo (funzionalità comune o appartenenza allo stesso contesto).
I packages sono utilizzati in Java per classificare, organizzare, evitare conflitti, controllare l'accesso/posizione a classi in modo più semplice. Un packageè, infatti, un meccanismo per organizzare le classi Java in gruppi logici, principalmente (ma non solo) allo scopo di definire namespace distinti per diversi contesti. Il package è un contenitore (cartella) per raggruppare definizioni di classi (o entità analoghe quali interfacce, enumerazioni...) logicamente correlate.

Tra i principali packages ci sono:
java.lang: è il package che contiene le classi nucleo del linguaggio, come System e String, funzionalità di base del linguaggio e tipi di dato fondamentali
java.util: raccoglie classi d'utilità, come Scanner, Date —classi di collezione (strutture dati)
java.io: contiene classi per realizzare l'input e l'output in Java java.awtcontiene classi per realizzare interfacce grafiche

Di default, in ogni file Java è importato automaticamente tutto il package java.lang, senza il quale non potremmo utilizzare classi fondamentali quali System e String. Notiamo che questa è una delle caratteristiche che rende Java definibile come "semplice". Per utilizzare una classe della libreria(e quindi tutti i metodiin essa definiti) , bisognerebbe specificarla attraverso il suo nome qualificato(esplicitando il percorsodove è definita tale classepotendone usaretutti i relativi metodi). La notazione "punto" è usata per distinguere le cartelle e sottocartelle del percorso fisico: java.lang significa la sottocartella lang all'interno della cartella principale dove è stata installata la piattaforma

Dal codice all'esecuzione

File "source.java" redatto con un editor od un IDE → **Source Code**

Comando: "javac source.java" per generazione byte code → **Java Compiler**

File source.class contenente il byte code eseguibile da JVM → **Byte Code**

Byte C

Comando: "java source.class" per esecuzione del byte code → **Java Virtual Machine**

Interprete bytecode

Istruzioni da JVM eseguite nel sistema operativo ospite → **Operating System**

Compilatore (JIT) Just In Time bytecode

Istruzioni in linguaggio macchina

Codice Java

```
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

Byte Code

```
0:  iconst_2
1:  istore_1
2:  iload_1
3:  sipush  1000
6:  if_icmpge       44
9:  iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge       31
16: iload_1
17: iload_2
18: irem
19: ifne    25
22: goto    38
25: iinc    2, 1
28: goto    11
31: getstatic       #84;
34: iload_1
35: invokevirtual   #85;
38: iinc    1, 1
41: goto    2
44: return
```

La classe Java: HelloWorld

Parola chiave **class** che definisce una classe in Java.

File sorgente "HelloWorld.java"

Modificatore di accesso della classe: indica che la classe è istanziabile nel metodo chiamante di altra classe (**public**).

HelloWorld: un sorgente .java può avere più classi ma solo una è **public** ed il suo nome coincide col nome del file sorgente.

Il modificatore di accesso **public** rende il metodo richiamabile dal codice Java esterno alla classe; in questo semplice esempio il chiamante è l'ambiente di esecuzione della JVM.

HelloWorld.java

```
public
class HelloWorld {

    public
    static
    void main (String[] args) {
        System.out.println("Hello World!");
    }

}
```

Modificatore di accesso **static**: indica che per eseguire il metodo della classe la stessa non deve essere istanziata, ossia non occorre definire ed assegnare una variabile di istanza rappresentante il riferimento all'oggetto della classe istanziata.
Quindi alla classe non deve essere associata una variabile d'istanza per eseguire il metodo.
Quando la JVM esegue il metodo **main** non alloca spazio nell'Heap per le variabili membro (variabili dello stato dell'oggetto)

Metodo **main**: indica alla JVM il metodo predefinito da eseguire lanciando il comando "java HelloWorld.java"

Nella classe **System** (package **java.lang**) usa lo standard di output **out** (il video) per stampare il messaggio "Hello World" (metodo **println**). Si noti che non è necessario indicare l'import **import java.lang;**

Parametri di input al metodo **main** (un array di stringhe, di lunghezza non specificata).