Java per Sistemi Mobili e Ubiqui

Java per Ubiquitous Computing

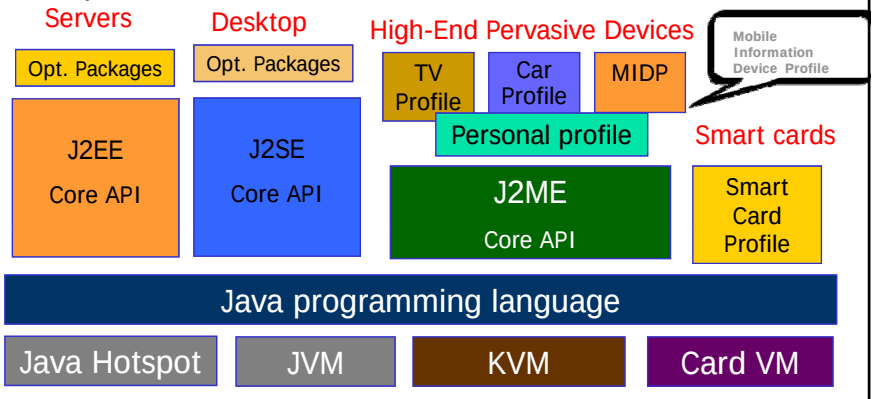
- Java Micro Edition
- Real-time Java
- KVM
- Waba
- J9
- JavaPhone
- Java Card

Java per Ubiquitous Computing

- Java Standard Edition non è adatto ad essere usato su dispositivi ubiqui con limitate risorse.
- Tuttavia, Java era stato inizialmente pensato sistemi di small computing.
- *Java Standard Edition* (J2SE): versione per PC tradizionali basato su JVM.
- *Java Enterprise Edition* (J2EE): versione per macchine server che include JSP, Java Beans, JDBC, …

Java per Ubiquitous Computing

- *Java Micro Edition* (J2ME): versione per sistemi ubiqui e pervasivi che include diverse macchine virtuali.



Java Micro Edition

- *Java 2 Micro Edition* (J2ME): versione per sistemi ubiui e pervasivi senza requisiti di elaborazione real-time.
  - Piccole memorie (128-512 KB)
  - Energia limitata (batterie)
  - Connessi in rete
  - Capacità grafiche limitate.

- Due configurazioni:
  - *Connected Device Configuration* (CDC)
  - *Connected, limited Device Configuration* (CLDC)

Java Micro Edition

- *Connected Device Configuration* (CDC)
  - Dispositivi con ROM > 512 KB e RAM > 256 KB
  - Libreria di user interface limitata.

- *Connected, limited Device Configuration* (CLDC)
  - Dispositivi con RAM tra 128 e 512 KB
  - Gestione messaggistica, sicurezza, wireless
  - KVM per questa configurazione.

- Anche: Embedded Java, Real-time Java, Java Card.

Java Micro Edition : Real-time Java

- Java Standard Edition non è adatta ad applicazioni real-time.
- Real-time Java : gestione di applicazioni hard real-time.
- *Tempi di esecuzione predicibili*
- *Schedulers adattabili*: schedulable objects e scheduler con comportamenti predicibili e cambio di scheduler.
- *Gestione avanzata della memoria* : per evitare garbage collection costosa.

Java Micro Edition : Real-time Java

- *Accesso alla memoria fisica:* per leggere o scrivere direttamente nella memoria di sensori/dispositivi.
- *Sincronizzazione di oggetti e thread* : code sincronizzate e monitor efficienti.
- *Gestione di eventi asincroni* : con trasferimento di controllo asincrono ad altri oggetti e thread.

Java Micro Edition : KVM

- *KVM* : Macchina virtuale Java per dispositivi pervasivi con memoria maggiore di 128 KB.
- Implementa le specifiche di JVM.
- Progettata per processori a 16 bit, funziona anche su processori a 32 bit.
- Uso di risorse minime, non massima performance.

Java Micro Edition : KVM

- Differenze con Java Standard Edition:
  1. Classi differenti per la User Interface: no Swing, no AWT, e uso di driver di I/O nativi.
  2. Alcune restrizioni: no RMI, thread grouping, array multidimensionali (opzionali).
  3. Sottoinsieme delle librerie J2SE: implementazione parziale di java.io e java.net.
- SDK per KVM. Disponibile su molti cellulari (Nokia, Sony, ..) e su diversi SO (es., Symbian).

WABA

- WABA è molto simile a Java ma non è la stessa cosa!
- WABA definisce un linguaggio, una VM, e un insieme di classi.
- Occorre usare WABA SDK e Java SDK nell'ambiente di sviluppo. La sintassi è identica, ma i linguaggi non sono compatibili.
- I programmi WABA possono essere compilati tramite il compilatore Java, ma poi il bytecode va tradotto nella WABA VM.

WABA

- WABA VM ha bisogno di una memoria molto limitata per eseguire applicazioni su dispositivi ubiqui: circa 60 KB.
- Possono essere aggiunte funzioni native alla WABA VM.
- Disponibile su Palm OS, Windows CE, BeOS, anche su Game Boy, iPaq.

IBM J9

- Visual Age Micro Edition J9: JVM implementata da IBM basata inizialmente su JDK 1.2.
- Sono disponibili varie configurazioni (class library):
  1. jclXtr : usa 92 KB di memoria ROM/flash.
  2. jclCore : usa 344 KB di memoria ROM/flash.
  3. jclGateway : usa 563 KB di memoria ROM/flash.
  4. jclMax : usa 2500 KB di memoria ROM/flash.

IBM J9

- Supporta code versioning e repository per lavoro in team di sviluppo.
- Smart Linker : elimina class file non usati.
- Ambiente di sviluppo disponibile su Windows e Linux.
- Piattaforme supportate: Windows CE, Embedded Linux, Palm OS, Neutrino.

Confronto : KVM, WABA, J9

	KVM	WABA	J9
Piattaforma	Palm OS, Windows CE, Symbian, Linux	Palm OS, BeOS, Windows CE	Palm OS, Windows CE, Neutrino, Linux
Prestazioni	Basse	Elevate	Elevate
Uso Memoria	Elevato	Basso	Basso
Librerie	Indipendenti dai disp.	Dipendenti dai disp.	Dipendenti dai disp.
Standard	J2ME		J2ME, Java
Codice Sorg.	Sun/Open Source	Open Source	No
Ambiente di Svil.	Java + KVM	Java + WABA VM	Ambiente + debug remoto

JavaPhone

- Una API di Java per telefoni cellulari supportata da Nokia, Sony Ericsson, Motorola, TI, Symbian.
- Obiettivo: sviluppo di servizi e applicazioni su dispositivi di telefonia mobile.
- API packages per
  - Direct Telephony Control, Address book and Calendar, User Profile, Network Datagram (SMS), Power Monitoring, Power Management, Application Installation, Communication, SSL.

JavaPhone

- Tramite queste API sono stati definiti due profili principali:
  - Internet Screenphone profile
  - Smart Phone profile
- Raggruppano diversi package (alcuni obbligatori, altri opzionali).
- In base ai dispositivi fisici si usano configurazioni Java differenti (con applet o senza).

Java Card

- Java Card è un ambiente per lo sviluppo di applicazioni su smart card (es., la SIM di un cellulare) in Java.
- Permette lo sviluppo di servizi e codice indipendenti dalla piattaforma e permette card multi-applicazione.
- Gli eseguibili di Java Card sono detti *Card applets*.
- Java Card deve essere supportato da un sistema operativo nativo, ma rende le applicazioni indipendenti da esso.

Java Card

- **Indipendenza dalla piattaforma**: un'applicazione per Java Card (Java Card Applet), scritta rispettando le regole imposte dall'API Java Card, può essere utilizzata senza modifiche su Java Card fornite da costruttori diversi (usando Card VM).

- **Supporto a più applicazioni**: su una stessa Java Card possono coesistere diverse applicazioni (Java Card Applet) indipendenti fra loro e selezionabili singolarmente in fase di esecuzione.

Java Card

- **Caricamento di nuove applicazioni dopo la consegna**: dopo che una Java Card è stata consegnata all'utente finale è ancora possibile caricare nuove applicazioni attraverso terminali addetti all'espletamento dei servizi.

- **Flessibilità**: il linguaggio utilizzato per programmare le Java Card è un subset del linguaggio Java: programmazione ad oggetti su smart card.

- **Compatibilità con gli standard delle SmartCard**: le Java Card sono compatibili con lo standard ISO 7816, lo standard più diffuso nel campo delle Smart Card.

Java Card

- Memoria **ROM**: è utilizzato per contenere il sistema operativo della JavaCard e la parte standard del JCRE (Java Card Runtime Environment) l'infrastruttura che permette il funzionamento della JavaCard.
- La dimensione di questo tipo di memoria è di circa 32 KByte per buona parte delle Card attualmente in commercio, mentre la dimensione minima richiesta è di 24 KByte;

Java Card

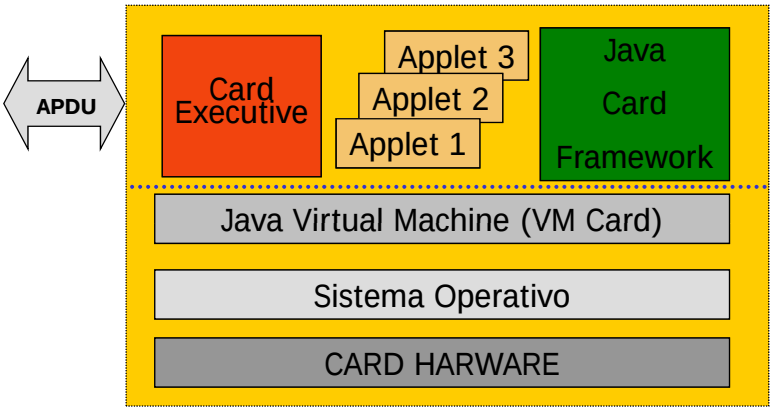
- Memoria **EPROM**: questo tipo di memoria, riscrivibile elettronicamente, è utilizzato per contenere le estensioni del JCRE e i Java Card Applet, nonché gli oggetti non temporanei creati durante l'esecuzione di Card Applet.
- La dimensione di questo tipo di memoria è di circa 16 KByte per buona parte delle Card attualmente in commercio, dimensione che rappresenta anche il limite minimo.

Java Card

- Memoria **RAM**: è utilizzato per contenere l'heap e lo stack necessari per l'esecuzione, quindi le variabili e gli oggetti temporanei creati durante l'esecuzione dei card Applet.
- La dimensione di questo tipo di memoria è di circa 1 KByte per buona parte delle Java Card attualmente in commercio, mentre la dimensione minima richiesta è di 500 Byte.

Java Card

- Il framework Java Card ha la seguente architettura:

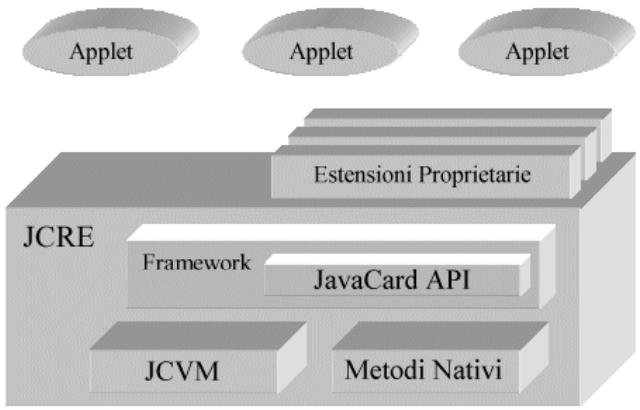


Java Card

- Il **Java Card Run Time Environment** (JCRE) ha le interfacce:

  1. **Card Executive** : gestisce la card, le sue applicazioni e le comunicazioni con l'esterno.
  2. La **JVM** che esegue i Card applet e le funzioni di libreria.
  3. **Metodi Nativi** per l'I/O e la sicurezza.
  4. Il **Java Card Framework** che fornisce le funzioni di libreria tramite 4 packages.

Java Card



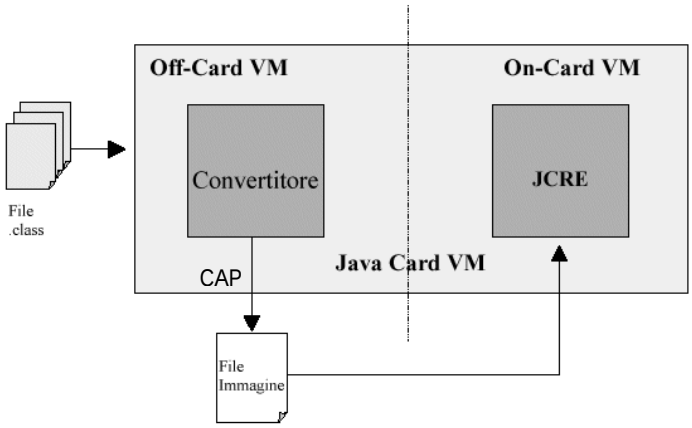
Java Card

- **Metodi Nativi**: metodi che forniscono le funzionalità di I/O, le funzionalità crittografiche e i servizi di allocazione della memoria.
- JCVM (Java **Card VM**): l'entità che si occupa dell'interpretazione del bytecode Java Card e che fornisce il supporto al linguaggio.
- **Java Card Framework**: classi che implementano l'API Java Card e include i package e le eventuali estensioni standard. Provvede alla distribuzione dei comandi provenienti dall'esterno agli altri componenti e all'installazione delle JavaCard Applet.

Java Card

- **Java Card API**: interfaccia utilizzata dai Java Card Applet per l'accesso al JCRE e ai Metodi Nativi. Questo componente fornisce quindi un insieme di classi che permette di accedere alle risorse standardizzate presenti sulla Java Card.
- **JCRE** (Java Card Runtime Environment): è l'ambiente JavaCard che permette la portabilità del codice scritto per una JavaCard su un'altra Java Card.
- **Estensioni Proprietarie**: classi aggiuntive definite dal costruttore della Java Card, che possono essere utilizzate dagli Applet.

Java Card : Due Ambienti



(in formato CAP- Converted APplet)

Java Card

- Il **Java Card Framework** consiste di 4 packages.
  1. java.lang : sottoinsieme di Java
  2. javacard framework : classi e interfacce per i card applet.
  3. javacard.security : classi e interfacce per la sicurezza della card.
  4. javacardx.cripto : classi e interfacce per la sicurezza di funzionalità esportate.

Java Card : applet su una smart card

- Gli applet installati su una smart card possono avere un tempo di vita pari a quello della card.
- Per terminarli bisogna de-installarli.
- Gli oggetti sono allocati nella EPROM e sono persistenti.
- Esistono array transienti, ma non oggetti transienti.
- Un applet installato può essere selezionato e usato da una applicazione esterna che invia all'applet i dati.

Java Card : applet su una smart card

LIMITAZIONI

- Il caricamento dinamico delle classi non è supportato.
- Non è possibile multithreading.
- La classe ***String*** non è disponibile.
- I tipi base ***char***, ***double***, ***float***, e ***long*** non sono implementati.
- Il tipo ***int*** è opzionale.

Sviluppo di un Java Card applet

Due passi principali:

1. Definire le APDU di comando e di risposta tra l'applet e l'applicazione host
2. Sviluppare l'applet.

Struttura tipica di un applet:

```
import javacard.framework.*
...
public class MioApplet extends Applet {
    // Definizioni di istruzioni per l'APDU
    ...
    MioApplet() {...}     // Construttore
    // Metodi per gestire l'esecuzione sulla card
    install() {...}
    select() {...}
    deselect() {...}
    process() {...}
    // Private methods
    ...
}
```


Java Card : applet su una smart card

```
import javacard.framework.*; // importazione di classi del framework
. . .

public class <Nome Java Card Applet> extends Applet {
. . .

 private <Nome JavaCard Applet>() {
  // metodo costruttore
  // si creano qui gli oggetti utilizzati dall'Applet
 }

 public static void install(APDU apdu) {
  // metodo invocato durante l'installazione dell'Applet
}
```

Java Card : applet su una smart card

```
public boolean select(APDU apdu) {
     // metodo invocato durante la selezione dell'Applet
    }

public void deselect() {
     // metodo invocato durante la deselezione dell'Applet
    }

public void process(APDU apdu) {
     // metodo invocato alla ricezione di un comando APDU
    }
. . .
}
```


Java Card : Metodi

```
public static void install(APDU apdu) {
     // metodo invocato durante l'installazione dell'Applet
}
```

- **`install()`**: questo metodo è richiamato dal JCRE come ultimo passo dell'installazione del JavaCard Applet.
- All'interno di questo metodo, il programmatore dovrebbe inserire tutte le istruzioni **`new`** di creazione degli oggetti che il Java Card Applet intende utilizzare durante l'esecuzione.
- Fra le istruzioni del blocco di codice relativo a questo metodo, il programmatore deve inserire anche una chiamata al metodo **`System.register()`** per la registrazione del Java Card Applet presso il JCRE.

Java Card : Metodi

```
public boolean select(APDU apdu)
                                {
   //  metodo  invocato  durante  la  selezione  dell'Applet
}
```

- **`select()`**: questo metodo è richiamato dal JCRE quando un Card Applet è selezionato (questo implica che il JCRE ha ricevuto un APDU riportante un'istruzione di selezione del Card Applet).
- Il Java Card Applet può accettare o rifiutare la selezione facendo restituire al metodo rispettivamente true o false.
- Notare che a questo metodo è passato come parametro l'APDU di selezione.

Java Card

```
public void deselect() {
   // metodo invocato durante la deselezione dell'Applet
   }
```

- **`deselect()`**: questo metodo è richiamato dal JCRE sul JavaCard Applet correntemente selezionato quando riceve un APDU contenente un'istruzione di selezione per deselezionare l'applet corrente.
- Questa chiamata precede immediatamente la chiamata della **`select()`** del JavaCard Applet cui la selezione fa riferimento.
- Questo è quello che accade anche se il Card Applet correntemente selezionato è lo stesso cui l'APDU di selezione fa riferimento.

Java Card

- ```
  public void process(APDU apdu) {
   // metodo invocato alla ricezione di un comando APDU
  }
  ```
- **`process()`**: questo metodo è richiamato la prima volta dal JCRE immediatamente dopo la chiamata di **`select()`**, ed ogni volta che arriva un APDU di comando dall'entità esterna, salvo che questo non sia un APDU di selezione per una JavaCard Applet installata o la richiesta di un metodo nativo.
- Dopo che sono state eseguite le operazioni, l'Applet risponde con un APDU di risposta, contenente gli eventuali dati di risposta e la Status Word di risposta. Quest'ultima informa l'entità esterna dello stato finale dell'esecuzione all'interno dell'Applet.

Java Card

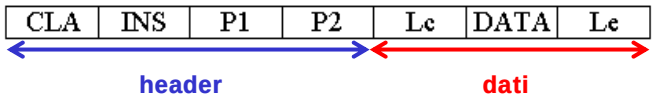
- Il mezzo che permette la comunicazione fra le Java Card ed i dispositivi esterni di controllo, che possono essere PC o workstation oppure dispositivi di controllo dedicati, prendono il nome di **CAD** (Card Acceptance Device).
- I CAD sono una gamma di dispositivi ampia
  - semplici lettori
  - dispositivi più complessi che presentano, ad esempio, anche un display e un tastierino numerico per l'inserimento di dati.
  ```

Java Card : APDU

- La comunicazione tra una smart card e l'esterno è realizzata tramite lo scambio di dati tramite lo standard **Application Protocol Data Unit**.
- Un APDU di comando è una sequenza di byte contenente le istruzioni inviabili dall'applicazione esterna alla Java Card e gli eventuali dati associati a queste istruzioni.
- La struttura degli APDU di comando è definita dallo standard ISO 7816 - parte 4.

Java Card : APDU

- Ogni **APDU di comando** può essere suddiviso in due parti, un **header** ed una parte **dati**:



- L'header è composto da quattro byte:
  - il CLA (byte di classe),
  - l' INS (byte di istruzione),
  - il P1 (primo byte del parametro)
  - il P2 (secondo byte del parametro).

CLA	INS	P1	P2

Java Card : APDU

- Il byte di classe **CLA** codifica, secondo lo standard ISO, la classe dell'istruzione.
- Il byte di istruzione **INS** codifica, sempre secondo lo standard ISO 7816 parte 4, l'istruzione che la Java Card deve eseguire.
- Il primo ed il secondo byte del parametro codificano invece il parametro a 16 bit relativo all'istruzione.
- La parte dati degli APDU di comando serve per comunicare eventuali dati al Card Applet destinatario dell'istruzione. Questa parte dati è a ***lunghezza variabile*** e può anche non essere presente.

Java Card : APDU

- La parte Dati è composta da:
  - **Lc** : un byte che indica la lunghezza dei dati inviati nell'APDU
  - **DATA** : i dati da scambiare (opzionali)
  - **Le** : un byte che indica la lunghezza dei dati attesi in risposta dell'APDU

Lc	DATA	Le

Java Card : APDU

- Ogni **APDU di risposta** è composto da i **dati** e da due **word di stato**

DATA	SW1	SW2

- I dati sono opzionali. La loro lunghezza è indicata nell'APDU di comando.
- I due byte di stato contengono informazioni di stato dell'operazione richiesta.