Software libero per programmare Java e Android (autore: Vittorio Albertoni)

Premessa

Il linguaggio Java non è dei più semplici: si può imparare comunque più facilmente del linguaggio C++ da cui deriva. Quasi nulla perdendo in termini di potenza, infatti, Java ci evita certe complicazioni legate all'uso del C++, come la gestione della memoria, l'uso dei puntatori, l'ereditarietà multipla.

Le librerie sono piene di ottimi testi che ci spiegano il linguaggio e su Internet, oltre a testi esplicativi, possiamo trovare molti tutorial che ci aiutano per esercitazioni pratiche.

Con il linguaggio Java possiamo creare qualsiasi tipo di programma per computer e, una volta creato e compilato su un computer dotato di un certo sistema operativo, questo programma potrà essere utilizzato anche su computer dotati di un sistema operativo diverso, purché vi sia caricata la macchina virtuale Java: la compilazione del programma Java, infatti, non produce un codice in linguaggio macchina che il sistema operativo dia in pasto alla CPU, ma un codice intermedio, detto bytecode, che sarà la piattaforma Java installata sul sistema operativo a dare in pasto alla CPU.

Una volta imparato il linguaggio Java, oltre ad avere la possibilità di sviluppare bellissime applicazioni per computer che gireranno indifferentemente su Windows, su Linux e su Mac OS X, saremo oltre la metà strada per sviluppare bellissime applicazioni (qui chiamate app) che gireranno indifferentemente su qualsiasi apparecchiature (tablet o telefonino) con sistema operativo Android. Dovremo solo acquisire, in più della conoscenza del linguaggio Java, la capacità di disegnare l'interfaccia grafica dell'applicazione: questa interfaccia sarà il vestito, la parte statica dell'applicazione e con il linguaggio Java la faremo vivere. Nel progettare l'interfaccia grafica stabiliremo dove sistemare sullo schermo, per esempio, un pulsante; con il linguaggio Java stabiliremo che cosa dovrà succedere nel momento in cui toccheremo quel pulsante.

Anche su come si costruiscano app per Android troviamo in libreria parecchio materiale; su Internet pure, magari un po' meno di quanto accada per Java.

Con questo mio piccolo contributo, pertanto, non parlerò di cose che altri, peraltro utilizzando spazi ben superiori a questo, hanno scritto meglio di quanto possa fare io e, data per acquisita la conoscenza del linguaggio Java e del linguaggio supplementare per le interfacce Android, mi limiterò qui a presentare alcuni strumenti con i quali applicare queste conoscenze con minore fatica per produrre ottimi risultati: ovviamente attingendo a ciò che ci regala il software libero.

Indice

1 Dotazione di base

Il minimo che dobbiamo avere sul nostro computer per sviluppare le applicazioni di cui stiamo parlando sono i kit di sviluppo.

Tra l'altro basterebbero questi per arrivare al risultato compiuto, anche se, soprattutto nel caso delle app per Android, la scelta di non ricorrere agli strumenti aggiuntivi che vedremo sarebbe un volersi fare del male da soli.

1.1 Il kit di sviluppo Java

Per sviluppare applicazioni Java serve il JDK, acronimo di Java Development Kit: in particolare, se parliamo di applicazioni per personal computer, il JDK di Java SE (Standard Edition).

Ne esistono praticamente due: quello open source protetto da licenza GNU-GPL, cioè software libero, si chiama OpenJDK; l'altro, protetto da licenza proprietaria Oracle, si chiama semplicemente JDK. Entrambi sono gratuiti.

Se vogliamo usare il JDK proprietario Oracle dobbiamo andare all'indirizzo

http://www.oracle.com/technetwork/java/javase/downloads/index.html

dove troviamo i pacchetti per i sistemi operativi Linux, Mac OS X, Solaris e Windows e tutte le istruzioni per l'installazione.

Se vogliamo usare OpenJDK dobbiamo andare all'indirizzo

http://openjdk.java.net/

dove la Oracle ci indica come installare il pacchetto sui sistemi Linux, praticamente dirottandoci sui repository delle varie distro.

Per poter installare OpenJDK su Mac OS X e Windows o su Linux bypassando la repository del distributore dobbiamo andare su GitHub, precisamente all'indirizzo

https://github.com/alexkasko/openjdk-unofficial-builds#openjdk-unofficial-installers-for-windows-linux-and-mac-os-x

dove, come si evince da quanto scritto nell'indirizzo stesso, troviamo codice binario definito "unofficial": provenendo da GitHub penso comunque si tratti di cose serie.

Una volta installato il JDK, soprattutto se il programma che vogliamo produrre è semplice e se abbiamo una conoscenza del linguaggio che non ci obblighi ad andare a scartabellare manuali per ogni riga di istruzioni, scriviamo in linguaggio Java la o le classi che costituiranno il programma utilizzando un qualsiasi editor di testo (non word processor ma editor di testo), la o le memorizziamo con l'estensione .java in una directory dedicata e, posizionandoci in questa directory, la o le compiliamo a terminale con il comando `javac` seguito dal nome del file .java con l'estensione: ci troveremo così, nella stessa directory, tanti file, con lo stesso nome del o dei file .java e l'estensione .class: questi ultimi, con estensione .class, sono i file in bytecode che costituiscono il programma, che potrà essere lanciato con il comando a terminale `java` seguito dal nome del file .class, senza estensione, che contiene il metodo `main`. Per creare un archivio .jar eseguibile possiamo inserire nella directory dove abbiamo i file .class un file di testo, che chiamiamo `manifest.txt`, con la sola riga `Main-Class: nome,`

dove al posto di nome mettiamo il nome, senza estensione, del file .class contenente il metodo `main` e, scritta la riga, diamo invio (è fondamentale che la riga termini con un accapo, cioè con un carattere di fine riga). A questo punto, posizionati nella solita directory, diamo il comando a terminale

```
jar cfm <nome.jar> manifest.txt *.class
```

dove <nome.jar> è il nome che vogliamo dare al file .jar che stiamo costruendo, e ci ritroveremo, sempre nella stessa directory, il nostro file .jar pronto per essere distribuito.

Il vantaggio del file .jar è quello di radunare in un solo archivio compresso tutto il codice che serve per eseguire il programma, in modo che programmi costruiti con più classi e più librerie non di sistema, siano concentrati in una sola sede.

Per eseguire il file .jar da terminale si dà il comando `java -jar` seguito dal nome del file .jar con estensione. Lo stesso comando, inserito in un launcher, ci consentirà di far partire il programma da dove vogliamo (da un menu, dal desktop, ecc.). Teniamo presente che, nel sistema Windows, i file .jar si eseguono anche semplicemente con doppio click sul loro nome.

1.2 Il kit di sviluppo Android

Per sviluppare app Android dobbiamo avere lo stesso kit di sviluppo Java visto nel paragrafo precedente e, in più, Android SDK, che sta per Android Software Development Kit.

All'indirizzo

http://developer.android.com/sdk/installing/index.html

ci viene offerta la doppia possibilità

- di installare Android Studio,
- di installare Stand-Alone SDK tools.

Il tutto gratuitamente e con riferimento a tutti i sistemi operativi Linux, Mac OS X e Windows.

L'installazione di Android Studio è l'autostrada. Ad Android Studio si è giunti in tempi relativamente recenti, dopo un periodo iniziale in cui veniva distribuito solo il kit Stand-Alone SDK tools e un periodo intermedio in cui veniva distribuito un pacchetto bundle che riuniva l'SDK e l'IDE Eclipse dotata del plugin ADT. L'attuale versione di Android studio è ancor più compatta ed è basata su un altro IDE, Intellij Idea, evidentemente ritenuto, forse non a torto, migliore di Eclipse.

Essendo un'autostrada non la si può percorrere con motorini: a cominciare dalla RAM, raccomandata nella dimensione di 4 GB ma che se è minore, più che farci lavorare, ci fa arrancare (la minima, indicata in 2 GB, fa ridere). Nel caso di Windows si richiede una versione da Vista in su (sia a 64 che a 32 bit). Nel caso di Mac si richiede OS da 10.8.5 in su. Nel caso di Linux si richiede un 64 bit su cui girino anche applicazioni a 32 bit. In ogni caso la risoluzione minima dello schermo è richiesta in 1280 x 800. Infine la versione minima del JDK è la 7.

Se non siamo così pretenziosi e non disdegniamo percorrere strade provinciali, vuoi perché abbiamo un mezzo modesto, come un vecchio Windows XP, un computer con 2 GB di RAM, ecc., vuoi perché ci piace di più un percorso vario, possiamo optare tranquillamente per

l'installazione del pacchetto Stand-Alone SDK tools e abbinarlo ad un IDE che ci dia una mano.

Ciò che è esclusa è la pretesa di arrivare ad un file .apk, cioè ad una app Android distribuibile, solo avvalendoci dell'Android SDK che troviamo nel pacchetto Stand-Alone SDK tools.

2 IDE di supporto

IDE sta per Integrated Development Environment.

L'IDE è un ambiente di sviluppo integrato che, in presenza della stesura di progetti complessi, costituiti da un insieme di file e di oggetti che debbano interagire tra loro per addivenire alla codifica di un programma eseguibile, rappresenta l'unica possibilità di arrivare in tempi ragionevoli ed evitando i più banali errori, ad un risultato accettabile.

Come ho detto nel capitolo precedente, se per lo sviluppo di applicazioni Java per computer l'IDE è estremamente utile, per lo sviluppo di app Android è assolutamente necessario (se facciamo ricorso ad Android Studio l'IDE ce lo ritroviamo integrato con l'Android SDK).

Insostituibile vantaggio che ci offre l'IDE per Java, almeno in tutti e tre i casi che presenterò qui di seguito, è quello della code completion, attraverso la quale, aiutati dal fatto che in Java tutto è un oggetto, non appena scriviamo il nome di una classe o di un oggetto che ci serve per realizzare la nostra riga di programma vediamo comparire l'elenco dei metodi contenuti in quella classe in modo da poter scegliere quello che ci serve, dopo di che vedremo l'elenco dei parametri che quel metodo richiede, ecc.: il tutto mettendoci al coperto da errori di sintassi dovuti ad incertezze sul nome esatto di un metodo, all'inserimento di parametri di tipo sbagliato, ecc. Infine, se la riga che abbiamo scritto è sbagliata o non ha senso nel linguaggio Java, la cosa ci verrà vistosamente segnalata con l'indicazione di ciò che non va.

Mantenendo fede alla destinazione hobbistico-dilettantistica di questi appunti e alla volontà di privilegiare software a licenza libera non parlerò di quel monumento che è Jdeveloper distribuito dalla Oracle con licenza proprietaria e richiamerò i tre più alla mano, ai quali, peraltro, non manca nulla che sia richiesto da professionisti.

Per non fare torti a nessuno li racconto in ordine di anzianità.

2.1 NetBeans

L'idea di progettare uno strumento per sviluppare con il neonato linguaggio Java matura nel 1996 ad opera di un gruppo di studenti della Facoltà di Matematica e Fisica all'Università Carolina di Praga. La Società nata per realizzare l'idea, prima ancora di licenziare un prodotto finito, fu acquisita dalla Sun Microsystem, la mamma del linguaggio Java, che lanciò la prima versione di NetBeans nel giugno del 2000.

Da Sun Microsystem si passa ovviamente a Oracle, che dirige tuttora la comunità di sviluppatori che lavorano attorno a NetBeans ed ai suoi plugin.

E' distribuito gratuitamente sotto licenza GPL2 e possiamo scaricarlo all'indirizzo

https://netbeans.org/,

dove lo troviamo, anche in lingua italiana, per i tre sistemi operativi Linux, Windows e OS X, in varie combinazioni: per programmare in Java e Android basta quella denominata Java SE, che è anche la più leggera.

Un avvertimento agli utenti Linux: il NetBeans installato da repository di Ubuntu a volte ha dato segni di malfunzionamento. E' pertanto buona cosa installare NetBeans ricorrendo alla fonte, che è quella indicata sopra.

Personalmente lo ritengo il migliore per programmare in Java, soprattutto se vogliamo sviluppare applicazioni dotate di interfaccia grafica, grazie alla facilità con cui ci consente di legare il disegno dell'interfaccia con il codice per darle vita (ciò che non fanno molto bene gli altri due IDE che vedremo).

Alcuni suggerimenti per le cose di meno immediato apprendimento.

Se vogliamo utilizzare librerie che non fanno parte della collezione ufficiale Java, per esempio predisposte da noi stessi (come la mia libreria per calcoli geometrici scaricabile leggendo l'articolo "Una libreria per calcoli geometrici" sul mio blog *www.vittal.it*) dobbiamo innanzi tutto catalogarla in NetBeans. Per farlo, dal menu STRUMENTI (TOOLS) -> LIBRERIE (LIBRARIES) apriamo la finestra di dialogo GESTORE DELLA LIBRERIA (ANT LIBRARY MANAGER), clicchiamo su NUOVA LIBRERIA (NEW LIBRARY) e seguiamo la procedura che si apre; se, oltre ad indicare l'indirizzo del file .jar di libreria, indichiamo nelle apposite cartelle anche gli indirizzi del source e del javadoc, potremo godere anche per la nostra libreria dei vantaggi della code completion. Una volta catalogata, la libreria è richiamabile nel progetto cliccando destro sul nodo LIBRERIE (LIBRARIES) nella scheda progetto e scegliendo AGGIUNGI LIBRERIA... (ADD LIBRARY...) nel menu a tendina: si apre così una finestrella con l'elenco delle librerie catalogate, nel quale possiamo scegliere quella che desideriamo utilizzare.

Terminato il progetto e sperimentato il risultato del nostro lavoro, clicchiamo destro sul primo nodo con il nome del progetto nella scheda di progetto, scegliamo PROPRIETÀ (PROPERTIES); nelle categorie elencate nella finestra di dialogo che compare scegliamo IMPACCHETTAMENTO (PACKAGING), spuntiamo tutte le tre opzioni nella successiva finestra di dialogo e clicchiamo su OK. Dopo di che diamo il BUILD (dal menu ESEGUI (RUN)) e, nella sottodirectory DIST della directory dove abbiamo memorizzato il progetto stesso, troviamo il file eseguibile .jar dell'applicazione. Se abbiamo usato librerie non di sistema, queste non sono inserite nel file .jar ma le troviamo a parte, sempre nella directory DIST. Per cui, se vogliamo distribuire la nostra applicazione, dobbiamo distribuire non solo il file .jar ma tutto il contenuto della directory DIST: il che può essere fastidioso.

Purtroppo, a differenza di quanto avviene per gli altri IDE che vedremo, NetBeans non inserisce, nemmeno su richiesta, le librerie terze, forse per evitarci il rischio di noie con i diritti d'autore e renderci ben consapevoli che, distribuendo librerie terze, stiamo distribuendo il frutto di lavoro non nostro. Possiamo tuttavia forzare NetBeans a farlo, seguendo questa procedura.

Apriamo la scheda FILE, tra quelle che vediamo nella colonna di sinistra dell'area di lavoro e, con doppio click sul file `build.xml`, lo portiamo nell'editor a destra. Scorriamo lo scritto e, in fondo, appena prima dell'ultima riga `</project>` inseriamo quanto segue:

```
<target name="impacchetta" depends="jar">
<property name="store.jar.name" value="<NOME_FILE_JAR>"/>
<property name="store.dir" value="store"/>
<property name="store.jar" value="${store.dir}/${store.jar.name}.jar"/>
<echo message="Packaging ${application.title} into a single JAR at ${store.jar}"/>
<delete dir="${store.dir}"/>
<mkdir dir="${store.dir}"/>
<jar destfile="${store.dir}/temp_final.jar" filesetmanifest="skip">
<zipgroupfileset dir="dist" includes="*.jar"/>
```

```
<zipgroupfileset dir="dist/lib" includes="*.jar"/>
<manifest>
<attribute name="Main-Class" value="${main.class}"/>
</manifest>
</jar>
<zip destfile="${store.jar}">
<zipfileset src="${store.dir}/temp_final.jar"
excludes="META-INF/*.SF, META-INF/*.DSA, META-INF/*.RSA"/>
</zip>
<delete file="${store.dir}/temp_final.jar"/>
</target>
```

con l'avvertenza di inserire, al posto di <NOME_FILE_JAR> il nome che vogliamo dare al file .jar da produrre, indicando anche l'estensione .jar.

A questo punto, memorizzato il file build.xml così rettificato, con click destro sul file build.xml stesso nella scheda FILE si apre un menu in cui puntiamo su ESEGUI OBIETTIVO (RUN TARGET) e, nel menu successivo, su ALTRI OBIETTIVI (OTHER TARGETS) e, finalmente, si clicca su IMPACCHETTA.

Ora, in una sottodirectory nominata STORE nella directory del nostro progetto, troviamo un file .jar che comprende anche la libreria.

La produzione del file .jar possiamo comunque affrontarla anche seguendo il metodo descritto a pagina 3, nel paragrafo 1.1, dedicato al JDK, adattando questo metodo a come la gestione del progetto fin qui effettuata da NetBeans ha archiviato e interconnesso le classi che compongono il progetto stesso.

In questo caso, terminato di scrivere il nostro progetto e sperimentatolo con un Run, seguendo il percorso `directory del progetto -> build -> classes -> nome_progetto` troviamo i file in bytecode della o delle classi che lo compongo, con l'estensione .class. Apriamo nella directory del progetto una sottodirectory che, per esempio, chiamiamo `jar` e al suo interno copiamo la directory <nome_progetto> contenente queste classi, inseriamo il file `manifest.txt` con l'indicazione del percorso alla classe contenente il metodo `main` (scrivendo `Main-Class:` <nome_progetto>/<nome_classe_con_il_main_senza_estensione> seguito da Invio) e la directory contenente il file .class della libreria: questa directory, con il relativo contenuto, la otteniamo estraendo il file .jar della libreria stessa.

Posizionati nella directory che abbiamo chiamato JAR diamo il comando

```
jar cfm <nome.jar> manifest.txt <directory_delle_classi> <directory_libreria>
```

e, nella stessa directory, troveremo l'eseguibile .jar da distribuire, contenente anche la libreria.

Bene sottolineare, comunque, che tutta questa manfrina nasce solo quando usiamo librerie esterne per il nostro progetto. In caso contrario le cose filano molto più lisce.

* * *

NetBeans, arricchito di un necessario plugin, si presta anche per la realizzazione di progetti Android, pur con il difetto che vedremo.

La procedura di installazione del plugin è la seguente e occorre attuarla con il collegamento Internet attivo e, ovviamente, con installato l'Android SDK di cui abbiamo parlato nel paragrafo 1.2.

Dal menu STRUMENTI (TOOLS) scegliamo PLUGIN e, nella finestra che comparirà, spostiamoci sulla scheda IMPOSTAZIONI (SETTINGS); qui clicchiamo sul pulsante AGGIUNGI (ADD). Nella finestrella che compare inseriamo nel campo NOME (NAME) un nome descrittivo, ad esempio `Android`, e nel campo URL il seguente indirizzo

`http://nbandroid.org/release72/updates/updates.xml`

spuntando la casella corrispondente all'opzione RILEVA AGGIORNAMENTI AUTOMATICAMENTE (CHECK FOR UPDATES AUTOMATICALLY); concludiamo con OK. Ora ci spostiamo sulla scheda PLUGIN DISPONIBILI (AVAILABLE PLUGINS), nell'elenco selezioniamo ANDROID e clicchiamo su INSTALLA (INSTALL). Confermiamo tutto quanto ci verrà chiesto di confermare durante l'installazione e, in pochi attimi, abbiamo finito. Ultima cosa: dal menu STRUMENTI (TOOLS) andiamo su OPZIONI (OPTIONS) e ci spostiamo sulla scheda VARIE (MISCELLANEOUS) dove troviamo una scheda ANDROID nella cui finestrella SDK LOCATION inseriamo il path all'Android SDK che abbiamo installato sul nostro computer. Al rilancio il nostro NetBeans è pronto per programmare Android.

Il difetto di NetBeans è che non ha un tool visuale per costruire l'interfaccia grafica dell'app, come avviene, invece, per gli altri due IDE che vedremo. Il file main.xml che contiene la descrizione di questa interfaccia grafica ce lo dobbiamo scrivere noi nell'editor in linguaggio xml.

L'inconveniente, se vogliamo evitare questa fatica, è superabile ricorrendo a un tool esterno che si chiama DroidDraw e che possiamo scaricare gratuitamente a questo indirizzo

http://www.softpedia.com/get/Programming/Coding-languages-Compilers/DroidDraw.shtml

Il tool è contenuto in un file zip dove troviamo un eseguibile java (droiddraw.jar) e un eseguibile Windows (droiddraw.exe). Purtroppo si tratta di un software vecchiotto e che non è stato più aggiornato. Con esso possiamo disegnare visualmente la nostra interfaccia e memorizzarla in un file descrittivo in formato main.xml che andremo ad inserire nel nostro progetto Android su NetBeans (scheda progetto, nodo RESOURCES -> LAYOUT) al posto di quello che c'è.

Completata la nostra programmazione diamo il BUILD (dal menu ESEGUI (RUN)) e, nella sottodirectory BIN della directory dove abbiamo memorizzato il progetto, troviamo il file .apk di debug che possiamo provare in un emulatore Android o su un vero apparecchio Android.

Per produrre il file .apk distribuibile anche su Google Play clicchiamo destro sul nodo principale del progetto nella scheda progetto (nella colonna sinistra dell'area di lavoro), scegliamo EXPORT SIGNED ANDROID PACKAGE dal menu a discesa che si apre e seguiamo le istruzioni.

* * *

Se a NetBeans darei un 9 più per programmare Java, gli darei un 6 meno per programmare Android.

2.2 Intellij Idea

Come l'idea di NetBeans è nata a Praga, anche la prima versione di Intellij Idea è stata rilasciata, nel gennaio del 2001, a Praga ad opera della Intellij, poi divenuta JetBrains, azienda di software che ha tuttora la sede principale a Praga.

Per le nostre esigenze basta e avanza l'edizione della community, gratuita e libera, distribuita sotto licenza Apache2; l'edizione Ultimate, che tratta molte altre cose oltre a Java e Android, non è libera, è rilasciata sotto licenza proprietaria e costa anche parecchio.

Il sito da cui possiamo scaricare il prodotto, disponibile per Linux, Mac OS X e Windows, è

https://www.jetbrains.com/

scegliendo la scheda IDEs.

L'installazione è semplicissima e una volta eseguita abbiamo a disposizione tutto ciò che ci serve per programmare in Java, pur con un designer visuale per le GUI non all'altezza di quello di NetBeans. Ovviamente dobbiamo avere installato il JDK.

Alcuni suggerimenti per le cose di meno immediato apprendimento.

Per aggiungere librerie terze che ci possano servire per il nostro progetto Java, da menu File scegliamo Project structure... e, dalla finestra di dialogo che compare, apriamo la scheda Libraries; clicchiamo sul pulsante +, scegliamo Java dal menu a discesa che si apre e, aiutati dal browser che ci si presenta, andiamo a scegliere il file .jar di libreria che ci serve nella posizione in cui è archiviato.

Terminato il progetto e sperimentato il risultato del nostro lavoro, per produrre l'eseguibile .jar, da menu File scegliamo Project structure... e, dalla finestra di dialogo che compare, apriamo la scheda Artifacts; clicchiamo sul pulsante + e nelle finestrelle che si aprono scegliamo Jar e poi From modules with dependencies...; nella finestrella Main Class della finestra di dialogo che si apre digitiamo il nome della classe che contiene il metodo `main`, senza estensione e, se vogliamo che il file .jar contenga le librerie esterne che abbiamo usato, spuntiamo l'opzione extract to the target JAR sotto la voce JAR files from libraries, infine diamo OK; si apre una finestra di riepilogo in fondo alla quale sono indicate le proprietà del manifest file e, nella finestrella Main class, introduciamo il nome della classe che contiene il metodo `main` scegliendolo dall'elenco che si apre cliccando sul pulsante con i puntini; ridiamo OK. Ora da menu Build scegliamo Build artifacts... e diamo il Build. All'interno della directory contenente il nostro progetto, nel percorso `out -> artifacts` troviamo il nostro eseguibile .jar contenente anche la libreria.

* * *

Una volta eseguita l'installazione, Intellij Idea è pronto anche per programmare Android senza bisogno di plugin. Ovviamente avendo installato anche Android SDK.

Personalmente lo ritengo il miglior IDE per programmare Android, e pare sia diventata di questo parere anche Oracle che, dopo anni di raccomandazioni per l'utilizzo di Eclipse, ha basato proprio su Intellij Idea il recente Android Studio.

In confronto con gli altri due IDE qui presentati è quello che richiede più risorse (almeno 1 GB di RAM essendo 2 GB la dimensione raccomandata).

Ottima la possibilità di costruire visualmente l'interfaccia grafica delle app.

Completata la nostra programmazione da menu Build scegliamo MakeProject e, nel percorso `out -> production -> nome_app` della directory dove abbiamo memorizzato il progetto, troviamo il file .apk di debug che possiamo provare in un emulatore Android o su un vero apparecchio Android.

Per produrre il file .apk distribuibile anche su Google Play da menu File scegliamo Project Structure... e, nella finestra di dialogo andiamo su Artifacts: clicchiamo sul pulsante + e, dal menu a caduta che si apre, scegliamo Android Application e poi From module <nome_applicazione>; si apre un'altra finestra nella quale andiamo a cliccare sulla linguetta Android per aprire finalmente la finestra nella quale predisponiamo il nostro file .apk firmato: cominciamo scegliendone il nome (finestrella Name), il percorso alla directory in cui memorizzarlo (finestrella Output directory), il tipo (finestrella Type, scegliendo

RELEASE SIGNED dal menu a caduta). Infine dobbiamo scegliere se utilizzare un key store già esistente o crearne uno nuovo, nel qual caso dovremo compilare la finestra di dialogo apposita. Inserite le password diamo OK. Ultimo passo: da menu BUILD scegliamo BUILD ARTIFACTS... e con i due click successivi memorizzeremo finalmente il nostro file .apk firmato nella directory che abbiamo scelto prima (di default lungo il percorso `out -> artifacts -> nome_app` della directory dove abbiamo memorizzato il progetto).

* * *

A Intellij Idea darei un 8 più per programmare in Java e un 9 per programmare Android.

2.3 Eclipse

Dopo le due glorie mitteleuropee rappresentate da NetBeans e Intellij Idea, entrambe nate a Praga, veniamo al terzo famosissimo IDE distribuito come software libero sotto i termini della Eclipse Public License dalla Eclipse Foundation, un consorzio costituito da grandi società come IBM, Borland, HP, Ericsson e da storici distributori Linux come Red Hat e SUSE.

Eclipse nasce con una prima versione nel novembre 2001 per essere un IDE tutto fare: basta fornirlo del plugin che serve.

Il sito sul quale possiamo procurarci Eclipse è

https://eclipse.org/

dove troviamo la descrizione di tutte le attività della Eclipse Foundation e una sezione di download che ci offre le versioni per Linux, Windows e Mac OS X: la versione che qui ci interessa è la Eclipse IDE for Java Developers.

L'installazione è semplicissima e una volta effettuata, ovviamente essendo installato anche il JDK, abbiamo quanto serve per programmare in Java, tuttavia senza il tool per progettare visualmente le GUI.

Questo tool è installabile andando all'indirizzo

https://eclipse.org/windowbuilder/

dove clicchiamo sul pulsante DOWNLOAD e, nella finestra che si apre, troviamo una tabella dalla quale scegliamo il link corrispondente alla versione di Eclipse che abbiamo installato; apriamo il link e ne copiamo l'indirizzo negli appunti. Apriamo Eclipse, dal menu HELP scegliamo INSTALL NEW SOFTWARE e, nella finestra che si apre, clicchiamo su ADD in corrispondenza alla finestrella WORK WITH e, nella successiva finestra di dialogo, come LOCATION, incolliamo l'indirizzo che abbiamo negli appunti. Indi seguiamo le istruzioni, tenendo presente che, di tutto ciò che viene proposto di installare, ciò che interessa veramente ai nostri fini è il Swing Designer.

A questo punto Eclipse è pronto per programmare Java, anche dotato del designer visuale per le GUI, tuttavia non di semplice utilizzo se paragonato a quello di NetBeans.

Alcuni suggerimenti per le cose di meno immediato apprendimento.

Per aggiungere librerie terze che ci possano servire per il nostro progetto Java, dobbiamo innanzi tutto catalogarle nelle USER LIBRARIES. Per fare questo da menu WINDOW scegliamo PREFERENCES e nella finestra che compare apriamo il nodo JAVA, poi il nodo BUILD PATH e scegliamo la voce USER LIBRARIES; con il pulsante NEW apriamo una finestrella nella

quale diamo un nome alla libreria che stiamo inserendo, nome che comparirà nella zona delle DEFINED USER LIBRARIES. Selezionato questo nome, con il pulsante ADD EXTERNAL JARS gli abbiniamo il percorso al file .jar di libreria e poi dovremmo avere l'avvertenza di abbinare anche il percorso al javadoc della libreria stessa. Una volta catalogate le nostre librerie esse saranno richiamabili nei vari progetti cliccando destro, nel Package Explorer sulla sinistra della finestra di lavoro di Eclipse, sul nodo principale del progetto e scegliendo BUILD PATH -> ADD LIBRARIES dai menu a discesa; nella finestra successiva scegliamo USER LIBRARY e, premuto il pulsante NEXT, vediamo l'elenco delle nostre librerie esterne catalogate, dal quale scegliamo quella che ci interessa.

Terminato il progetto e sperimentato il risultato del nostro lavoro, per produrre l'eseguibile .jar, da menu FILE scegliamo EXPORT, apriamo il nodo JAVA che vediamo nella finestra che si apre e scegliamo la voce RUNNABLE JAR FILE: cliccando su NEXT apriamo la finestra successiva, nella quale possiamo scegliere il nome del file .jar da produrre, il percorso per la sua memorizzazione sul computer e se esso debba o meno contenere le librerie esterne eventualmente utilizzate nel progetto.

* * *

L'uso di Eclipse per programmare Android lo troviamo descritto in tutti i testi relativi alla programmazione Android fino alla comparsa di Android Studio: Eclipse è infatti sempre stato l'IDE raccomandato da Google.

Per programmare Android Eclipse ha bisogno del relativo plugin, per installare il quale apriamo Eclipse, dal menu HELP scegliamo INSTALL NEW SOFTWARE e, nella finestra che si apre, clicchiamo su ADD in corrispondenza alla finestrella WORK WITH e, nella successiva finestra di dialogo, come LOCATION, inseriamo l'indirizzo

`https://dl-ssl.google.com/android/eclipse/`

indi seguiamo le istruzioni.

Eclipse così attrezzato ci dà la possibilità di sviluppare app Android anche avvalendoci di un buon costruttore visuale della GUI.

Finito il nostro lavoro di programmazione da menu PROJECT scegliamo BUILD ALL e poi facciamo partire l'app cliccando sul pulsante RUN o scegliendo RUN dal menu RUN: con questo verrà predisposto un file .apk dell'applicazione stessa nella sottodirectory `bin` della directory dove abbiamo memorizzato il progetto.

Per produrre il file .apk firmato distribuibile andiamo sul menu FILE e scegliamo EXPORT: compare la finestra EXPORT nella quale apriamo il nodo ANDROID e scegliamo EXPORT ANDROID APPLICATION; con NEXT apriamo un'altra finestra nella quale dobbiamo scegliere il progetto dal quale vogliamo generare il file .apk (con il pulsante BROWSE... facciamo comparire l'elenco dei progetti candidati e scegliamo quello che ci interessa); con NEXT apriamo la finestra nella quale dobbiamo scegliere se avvalerci di una chiave già pronta o se dobbiamo costruirla e seguendo le istruzioni arriviamo finalmente al nostro file .apk firmato che verrà archiviato dove noi stessi abbiamo scelto di farlo.

* * *

A Eclipse darei un 8, sia per la programmazione in Java sia per la programmazione Android.