ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Backport di una applicazione da Java 8 a Java 7

Relazione finale in Programmazione ad Oggetti

Relatore

Prof. Mirko Viroli

Correlatore

Ing. Danilo Pianini

Presentata da

Giovanni Romio

SESSIONE III
ANNO ACCADEMICO 2014 - 2015

INDICE

1 INTRODUZIONE *5*

2 BACKGROUND *7*

2.1 Android, backporting e Java **7**

2.2 Gradle **9**

2.3 Protelis **13**

2.4 Java 8: Lambda e Stream **14**

2.5 Incompatibilità Java 8 e Android **16**

3 ANALISI *19*

3.1 Problemi e requisiti **19**

3.2 Streamsupport **20**

3.3 Retrolambda **23**

3.3.1 Descrizione funzionalità 23

3.3.2 JVM Invoke 24

3.3.3 Traduzione lambda-expression 25

3.3.4 Esempio di Retrolambda da linea di comando 26

3.3.5 Retrolambda con Streamsupport 28

3.4 Creazione di un'estensione del compilatore **29**

4 PROGETTO E IMPLEMENTAZIONE *31*

4.1 Integrazione degli strumenti di backport **31**

4.1.1 Retrolambda con Gradle 32

4.1.2 Retrolambda con Streamsupport e Gradle 33

4.2 Backport Protelis **34**

4.2.1 Integrazione di Streamsupport 35

4.2.2 Tabella delle conversioni 36

4.2.3 Conversione dei metodi statici nelle interfacce 38

4.2.4 Integrazione di Retrolambda 39

4.2.5 Verifica dell'effettivo funzionamento di Java 7 40

CONCLUSIONI *43*

BIBLIOGRAFIA *45*

1 INTRODUZIONE

In Java 8, ultimo aggiornamento ufficiale del linguaggio Java, sono state introdotte alcune nuove funzionalità che permettono l'integrazione di alcuni meccanismi legati ai linguaggi dinamici o funzionali, come le espressioni lambda, l'utilizzo degli stream e la dichiarazione di metodi statici all'interno di interfacce.

Android, per eseguire le applicazioni, fa affidamento a una macchina virtuale in grado di interpretare ed eseguire un bytecode corrispondente solamente alla versione Java 7.

Se si volesse installare un'applicazione scritta in Java 8 su Android, Dalvik VM, la JVM in esso presente, fallirà il processo di traduzione del bytecode. In questa tesi quindi esplorerò, sia ad alto livello che a basso livello, l'origine del problema e presenterò una soluzione di backporting per un'applicazione esistente.

L'applicazione in questione si chiama Protelis, integrata, scritta e basata sull'architettura Java che permette di produrre del codice resiliente per applicazioni destinate alla comunicazione fra grandi quantità di dispositivi mobili e uno o più server.

Per risolvere il problema di backport mi sono avvalso di alcuni strumenti come Gradle, Retrolambda e StreamSupport e ho quindi evidenziato un metodo progressivo, per fasi, seguendo il quale è possibile eseguire un'applicazione scritta e compilata in Java 8 su Java 7 e poterla in futuro utlizzare da Android.

2 BACKGROUND

2.1 Android, backporting e Java

Qualora un'applicazione scritta in Java 8 fosse pensata per essere eseguita da un dispositivo Android, bisogna produrre una sua versione adatta per tale environment. Su Android vi è, infatti, un runtime Java 7 che non supporta gli stream, non permette l'utilizzo delle lambda-expression, non supporta i default-methods e i metodi statici nelle interfacce.

Android è un sistema operativo per dispositivi mobili sviluppato da Google Inc. che è basato su kernel Linux. Esso non contiene codice GNU, pertanto non è da considerarsi una distribuzione GNU/Linux per sistemi embedded.
Android è stato progettato principalmente per smartphone e tablet, con interfacce utente specializzate tipo Android TV, Android Auto, Android Wear, Google Glass. Esso è, per la quasi totalità Free and Open Source Software, eccetto che per i driver non-liberi inclusi per i produttori di dispositivi e di alcune Google apps incluse, come il Google Play store.
Android è distribuito sotto i termini della licenza libera Apache 2.0 [1].

Il backporting indica l'azione di prendere una certa modifica software, per esempio una patch e applicarla ad una versione del software precedente a quella per cui è stata introdotta la modifica.
Il backporting è parte del processo di manutenzione nel ciclo di vita del software.
Per tradurre un software da Java 8 a Java 7 si può effettuare il backporting utilizzando alcuni strumenti open-source sviluppati dalla community.

Il metodo ottimale per produrre una versione del software corretta si divide in 3 fasi e segue un processo incrementale. Al termine di ciascuna fase si effettua la con il compilatore.

Il processo di backporting può essere schematizzato in:

- identificazione del problema nella vecchia versione del software che necessità di essere corretto da un backport,
- trovare quale modifica del codice risolve il problema,
- adattare la modifica al vecchio codice.

Normalmente, se le modifiche sono molte, queste vengono raggruppare in una patch che passa attraverso uno o più controlli di qualità. Le modifiche che riguardano un singolo aspetto del software sono semplici se solo poche linee di codice sono state cambiate, sono pesanti e invasive se riguardano diversi aspetti, per esempio molte modifiche in molteplici file. In quest'ultimo caso il backporting è spesso noioso e inefficiente e dovrebbe essere fatto solo se la vecchia versione del software è veramente necessaria e non può essere sostituita con una più recente.
I backport usualmente sono prodotti dal team che ha sviluppato il software, specialmente nel caso in cui il software in questione non è open-source, perché è necessario l'accesso al sorgente del software. Tali backport sono normalmente incorporati negli aggiornamenti delle vecchie versioni del software [2].

Uno dei principi fondamentali del linguaggio è espresso dal motto write once run anywhere (WORA): il codice compilato che viene eseguito su una piattaforma non deve essere ricompilato per essere eseguito su una piattaforma diversa. Il prodotto della compilazione è infatti in un formato chiamato bytecode che può essere eseguito da una qualunque implementazione di un processore virtuale detto Java Virtual Machine.

Java risulta essere uno dei linguaggi di programmazione più usati al mondo, specialmente per applicazioni client-server, con un numero di sviluppatori stimato intorno ai 9 milioni.
Il linguaggio fu originariamente sviluppato da James Gosling presso la Sun Microsystems che nel 2010 è stata acquisita dalla Oracle Corporation, attualmente detentrice del marchio registrato.

Il linguaggio deriva gran parte della sua sintassi dai linguaggi Simula, C e C++, ma ha meno costrutti a basso livello e implementa in modo più puro, rispetto al C++), il paradigma object-oriented [3].
Java 8 è la versione più recente del software Java, dotata di nuove funzionalità, miglioramenti e correzioni di errori che consentono di sviluppare ed eseguire i programmi Java in modo più efficiente. Le nuove versioni del software Java, prima di essere messe disponibili sul sito web www.java.com, per il download da parte degli utenti finali, vengono per prima cosa sempre rese disponibili agli sviluppatori, in modo che abbiano tempo a sufficienza per i test e le certificazioni [4].

2.2 Gradle

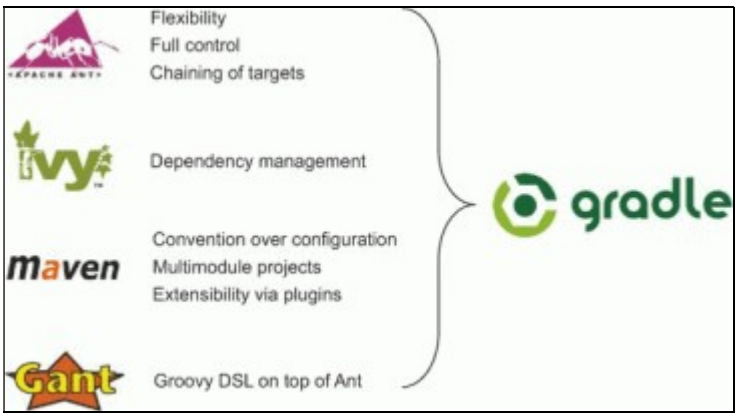
Gradle è un sistema open-source che permette di automatizzare alcune procedure legate allo sviluppo di un software. E' fondato sulle idee di Apache Ant e Apache Maven, introduce un domain-specific language (DSL) basato su Groovy, al posto della modalità XML usata da Apache Maven per dichiarare la configurazione del progetto. Gli script Gradle possono essere eseguiti direttamente, in contrasto con le definizioni dei progetti Apache Maven (pom.xml).
Al contrario di Apache Maven, che definisce il ciclo di vita di un processo, e di Apache Ant, dove l'ordine dei compiti, detti target, è determinato dalle dipendenze (depends on), Gradle utilizza un grafo aciclico diretto (DAG) per determinare l'ordine in cui i processi possono essere eseguiti.

Gradle è stato progettato per sviluppi multi-progetto e supporta sviluppi incrementali determinando in modo intelligente quali parti del build tree sono aggiornate (up-to-date), in modo che tutti i processi che dipendono solo da quelle parti non avranno bisogno di essere ri-eseguiti; così facendo, il software riduce significativamente il tempo di costruzione del progetto, in quanto, durante il nuovo tentativo di costruzione, verranno eseguite solo le attività il cui codice è

effettivamente stato alterato a partire dall'ultima costruzione completata. Gradle supporta anche la costruzione del progetto per processi concorrenti, il che consente di svolgere alcuni compiti durante la costruzione (ad esempio i test automatizzati attraverso gli unit test), eseguiti in parallelo su più core della medesima CPU, su più CPU o su più computer.

I plugin iniziali sono concentrati soprattutto sullo sviluppo e implementazione di Java, Groovy, Scala e C++, ma l'intenzione è quella di estendere il progetto anche ad altri linguaggi [5].

Gli strumenti integrati in Gradle sono: Ant, Ivy, Maven, Gant :



Uno script di build è composto da zero o più dichiarazioni di blocchi di script.
Le dichiarazioni possono includere metodo di chiamate, assegnazioni di proprietà, e delle definizioni di variabili locali.

I blocchi di scripts di livello superiore sono elencati di seguito:

artifacts { }

Configura gli elementi (artifacts) pubblicati per questo progetto.

buildscript { }

Configura il classpath* di build (configurazione) per questo progetto.

configurations { }

Definisce le configurazioni delle dipendenze (dependencies) per questo progetto.

dependencies { }

Configura le dipendenze per questo progetto.

repositories { }

Configura i repository per questo progetto.

Uno script di build (impostazioni) è anche uno script Groovy, e quindi può contenere quegli elementi consentiti in uno script Groovy, quali le definizioni dei metodi e le definizioni di classe [6].

Una task rappresenta un pezzo unico ed atomico di lavoro per una build, come ad esempio la compilazione di classi o la generazione di javadoc:

```
task myTask
task myTask { configure closure }
task myType << { task action }
task myTask(type: SomeType)
task myTask(type: SomeType) { configure closure }
```

Ogni attività fa parte di un progetto . È possibile utilizzare i vari metodi su TaskContainer per creare e istanziare alcune attività di ricerca . Ad esempio, TaskContainer.create ( ) crea un compito vuota con il nome dato. Ogni attività ha un nome che può essere utilizzato per fare riferimento al suo compito all'interno del progetto. Per ogni attività è previsto un percorso completo, univoco all'interno del progetto. Il percorso è la concatenazione tra il progetto possessore e il nome del compito. Gli elementi di percorso sono separati usando il carattere ":" [7].

```
task(hello) << {
    println "hello"
}

task(copy, type: Copy) {
    from(file('srcDir'))
    into(buildDir)
}
```

Gradle permette di definire dei default task che vengono lanciati se non viene specificato nessun altro task [8].

```
defaultTasks 'clean', 'run'

task clean << {
    println 'Default Cleaning!'
}

task run << {
    println 'Default Running!'
}

task other << {
    println "I'm not a default task!"
}
```

Un artefatto è uno dei molti tipi di sottoprodotti ottenuti durante lo sviluppo del software. Alcuni artefatti contribuiscono a descrivere la funzione, l'architettura e la progettazione di software (diagramma dei casi d'uso, diagrammi di classe, requisiti e documenti di progettazione).
Gli artefatti sono significativi dal punto di vista di gestione del progetto come prodotto finale. I risultati finali di un progetto software sono l'insieme dei suoi artefatti con l'aggiunta del software stesso .
Gradle permette di definire gli artefatti che verranno pubblicati insieme al progetto. In Gradle vengono definiti artefatti del progetto i file messi a disposizione al mondo esterno. Un esempio potrebbe essere una libreria, una distruzione ZIP o ciascun altro file. Ad un progetto possono essere associati molteplici artefatti.

Per definire un artefatto, contenente il progetto, in gradle si utilizza un archive-task [9]:

```
task myJar(type: Jar)

artifacts {
    archives myJar
}
```

2.3 Protelis

Lo sviluppo di sistemi di rete è davvero difficile. Per fare un buon sistema resiliente, è generalmente necessario il supporto contemporaneo di tre tipi di competenze nello stesso codice:

- Conoscenza generale per quanto riguarda lo scopo effettivo del sistema;
- Competenza di networking, per gestire le interazioni tra singoli dispositivi;
- Competenza in algoritmi distribuiti, per garantire che il comportamento collettivo sia auspicabile e resistente a tutti i tipi di errori e modifiche di sistema [10].

L'obiettivo del linguaggio Protelis è quello di rendere i sistemi collegati in rete resilienti e altrettanto facili da costruire, sia per reti complesse ed eterogenee sia per singole macchine e sistemi in cloud. Questo si ottiene separando i diversi compiti e facendo alcune delle parti più difficili e specifiche in maniera automatica ed implicita.

Alcune delle decisioni di progettazione chiave di Protelis sono:

- Esistono sono molti per rompere un sistema distribuito, il fatto che Protilis sia un linguaggio e non una libreria, consente di trattare implicitamente, in questo modo vengono ridotte notevolmente le possibilità di sbagliare.
- Protelis è ospitato e integrato con Java. E' stato scelto questo linguaggio di sviluppo per la sua particolare architettura molto leggera e per la grande dimensioni dell'ecosistema preesistente di infrastrutture e librerie per esso disponibili.
- Protelis è molto simile a Java per cui è facile da imparare e adottare.
- Protelis assicura una struttura sicura e resiliente, il suo nucleo è costituito da un modello teorico di programmazione aggregata molto simile alla programmazione funzionale delle lambda [10].

2.4 Java 8: Lambda e Stream

Di seguito viene fornito un breve riepilogo dei miglioramenti introdotti nella release Java 8:

- Espressione Lambda e supporto agli Stream
- API Data e Ora: questa nuova API permette agli sviluppatori di gestire la data e l'ora in una modalità più naturale, chiara e facile.
- Motore JavaScript Nashhorn: una nuova implementazione leggera e dalle elevate prestazioni del motore JavaScript è integrata in JDK ed è disponibile per le applicazioni Java mediante le API esistenti.
- Maggiore sicurezza: sostituzione della precedente lista gestita a mano dei metodi sensibili al chiamante con un meccanismo che identifica in modo preciso tali metodi e permette di rilevare l'affidabilità dei relativi chiamanti [11].

Le espressioni Lambda sono state introdotte per risolvere il problema della voluminosità delle classi anonime. Tramite l'utilizzo della corretta sintassi infatti è possibile convertire cinque linee di codice in una singola istruzione. La semplice soluzione orizzontale permette di risolvere il "problema verticale" presente nelle classi interne.
Un Lambda-expression è composta dalla lista argomenti, dalla freccia e dal corpo [12]:

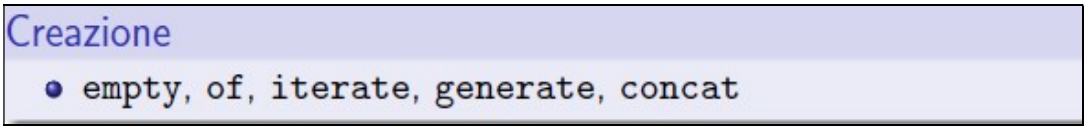
Lista argomenti	**Freccia**	**Corpo**
(int x, int y)	->	x + y;

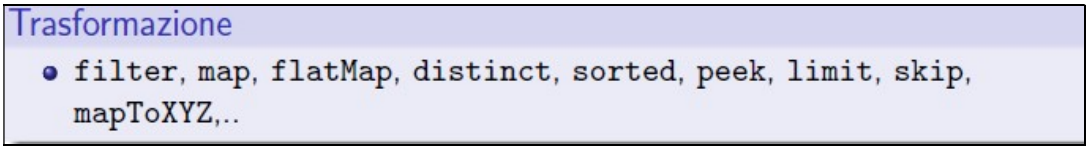
Uno Stream rappresenta un flusso sequenziale, anche infinito, di dati omogenei, usabile una volta sola, dal quale si vuole ottenere una informazione complessiva e/o aggregata.

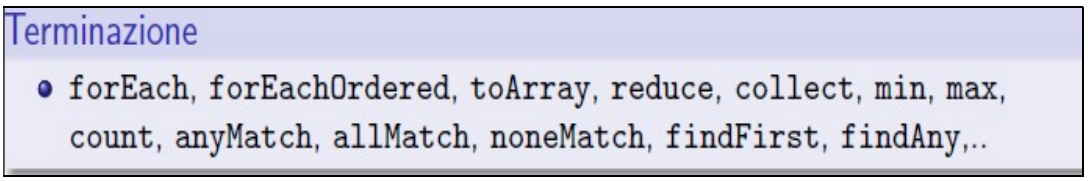
Lo Stream assomiglia al concetto di Iteratore, ma e più dichiarativo, perché non richiede una specifica imperativa per essere letto, e quindi é concettualmente più astratto.

Uno Stream manipola i suoi elementi in modo “lazy" (ritardato): i dati vengono processati mano a mano che servono, non sono memorizzati tutti assieme come nelle Collection.
E' possibile creare “catene" di Stream, implementate con decorazioni Successive, in modo funzionale, per ottenere flessibilmente computazioni non banali dei loro elementi. Questa modalità di lavoro rende le computazioni automaticamente parallelizzabili, ossia computabili da un set arbitrario di Thread [13].

Per creare uno stream bisogna definire le tre parti fondamentali, ovvero creazione, trasformazione, terminazione:







Tutte le collezioni a partire da Java 8 implementano i metodi citati, per cui usando una qualunque Collection è possibile generarne uno stream.

```
public interface Collection<E> extends Iterable<E> {

    ..

    Iterator<E> iterator();

    default boolean removeIf(Predicate<? super E> filter) {..}

    default Spliterator<E> spliterator() {..}

    default Stream<E> stream() {..}

    default Stream<E> parallelStream() {..}
}
```

2.5 Incompatibilità Java 8 e Android

L'obiettivo primario dei progettisti di Java è stato quello di sostenere i linguaggi dinamici, per esempio i linguaggi in cui non si sa alcuna informazione di tipo fino a quando il programma è eseguito.
In questo contesto, nella Java Virtual Machine, per generare efficacemente bytecode bisogna conoscere gli effettivi tipi dei dati e, per affrontare questo problema di ritardi, è stato introdotto il nuovo operando InvokeDynamic.
InvokeDynamic non rende definitiva l'invocazione vera e propria, ma una volta invocata, si aspetta che sia il software "supervisore" ad analizzare i parametri di chiamata attuali e generi la sequenza di trasformazione dei MethodHandle per abbinare in modo efficace i tipi degli argomenti dei metodi.

Il compilatore di Java 8, rispetto al precedente, porta a indubbi benefici, anche se integrare linguaggi diversi sulla macchina virtuale HotSpot determina un certo tipo di complessità.

Quando era in corso di progettazione InvokeDynamic, ci si rese conto che la cosa più problematica era quella di inviare correttamente ed efficacemente le chiamate dei metodi.
Non tutti i linguaggi utilizzano le regole di Java, alcuni linguaggi supportano le conversioni di tipo e di argomenti impliciti, altri possono modificare dinamicamente il bersaglio o le strategie di spedizione.

Il problema principale con InvokeDynamic è il fatto che ha una struttura dinamica.
Java è un linguaggio statico e tipizzato: tutti i tipi delle variabili, dei metodi e dei parametri sono noti al compilatore prima che emetta il bytecode.
Il compilatore JavaC (da JDK8) traduce le lambda-expression e risolve le chiamate ai metodi con invokeDynamic, per cui funziona senza i tipi delle informazioni.

La perdita inutile di tipi è problematico per le macchine virtuali che si suppongono per l'esecuzione in environemnt ristretti. Quando si è in un environemnt di esecuzione ristretto, non possiamo consumare le risorse, cercando di creare nuove classi. Ciò in fase di compilazione può essere troppo costoso, risulta molto più facile generare il formato corretto per l'esecuzione ahead-of-time [14].

3 ANALISI

3.1 Problemi e requisiti

Un'applicazione scritta in Java 8 non può essere eseguita su un dispositivo Android che supporta solo codice scritto e compilato in Java 7.
Nel dettaglio, l'odierna macchina virtuale Dalvik, che interpreta le istruzioni bytecode delle applicazioni non supporta l'utilizzo delle lambda, delle librerie introdotte con Java 8 né l'utilizzo di metodi statici all'interno delle interfacce.

Portare un'applicazione scritta in Java 8 ad Android significa, per prima cosa, rendere il bytecode della nostra applicazione, generato dal compilatore, compatibile con la JVM Dalvik. In seguito si dovrà progettare una struttura con un Manifest.xml, le Activity, i Layout e le resources.

I problemi di compatibilità sono riconducibili alle innovazioni di Java 8 nelle librerie, nel linguaggio e nel bytecode:

Librerie

Protelis è un linguaggio che utilizza un modello di programmazione aggregata e sfrutta alcune delle librerie introdotte con Java 8: java.util.stream, java.util.Optional, java.util.function. Il problema è quindi la ricerca di una libreria compilabile con Java 7 che sostituisca le nuove librerie sopracitate.

Linguaggi

I linguaggi di programmazione aggregata sono rafforzati notevolmente dalle lambda-expression introdotte con Java 8.

Protelis sfrutta a pieno le nuove API grazie all'utilizzo delle lambda-expression che non sono però compatibili con Java 7.

Nell'ultima versione di Protelis sono presenti alcune interfacce, legate alla definizione di nuovi tipi, che implementano alcuni metodi statici al proprio interno.

Questa tecnica può essere utilizzata sia per rendere il codice più pulito sia per evitare che avvenga un'implementazione sbagliata di un metodo presente in un'interfaccia presente in una classe figlia.

Bytecode

Il bytecode di un'applicazione prodotto da un compilatore Java , per essere utilizzato direttamente, deve utilizzare le librerie, le espressioni e le funzioni che sono compatibili con la stessa versione dell'interprete presente all'interno della JVM.

3.2 Streamsupport

StreamSupport è un backport delle API java.util.function e java.util.stream introdotte con Java 8 per un ambienta Java 6 o Java 7.

A causa della mancanza degli static-methods e dei default-methods nelle interfaccie in Java 7, la libreria è stata leggermente modificata in questa sezione ma nonostante tutto copre tutte le funzionalità introdotte con Java 8. Nel dettaglio gli static-methods e i default-methods nelle interfaccie sono stati spostati su classi compagne nello stesso package con il nome identico, ma con una "s" aggiunta (es. Comparator -> Comparators)

Per facilità d'uso i default-method, per la maggior parte delle interfacce funzionali, non sono stati mantenuti come metodi astratti in interfacce ridefiniti, mantenendo le interfacce metodo unico.
I default-methods mancant e statici si trovano nella classe compagna corrispondente [15].

Utilizzando l'IDE di sviluppo Eclipse, con lo scopo di eseguire alcuni test, è possibile configurare l'ambiente di sviluppo per Java 7.

Siccome in Eclipse possono essere installate più versioni di Java Developement Kit, è possibile cambiare a piacimento il livello del compilatore tra la versione 1.7 e la versione 1.8 .

Oltre alla versione del compilatore, si può decidere quali librerie di Java Standard Edition utilizzare nel progetto. Per testare il progetto quindi togliamo il riferimento alle API di JSE 8 così che si puà essere sicuri che gli stream di java.util.stream vengono segnalati come errori sia dall'IDE, così da facilitare la traduzione del codice, sia dal compilatore.

Un esempio indicativo è quello della generazione di uno stream relativo ad una lista, in questo caso di numeri interi, con l'iterazione dei suoi elementi visualizzandone il risultato:

```
package testStream;

import java.util.Arrays;
import java.util.List;


public class Main {

    public static void main(String[] args) {
        List < Integer > li = Arrays . asList (10 ,20 ,30 ,5 ,6 ,7 ,10 ,20 ,
        System . out . print (" Collection : ");
        li. stream (). forEach (i-> System . out. print (" "+i));
    }

}
```

Lo stesso risultato si può riprodurre tramite l'utilizzo di Streamsupport.
Si tratta di creare uno stream da una collezione e iterare i suoi elementi utilizzando il metodo statico presente nella libreria java8.util.stream.StreamSupport. In questo esempio lo stream prende come parametro in ingesso una lista di oggetti e per ciascun valore ne visualizza il risultato.

Se in questo caso venisse rimpiazzata anche la lambda-expression il codice diventerebbe compatibile con JDK7:

```
package testStream;


import java.util.Arrays;
import java.util.List;

import static java8.util.stream.StreamSupport.stream;


public class Main {

    public static void main(String[] args) {
        List < Integer > li = Arrays . asList (10 ,20 ,30 ,5 ,6 ,7 ,10 ,20 ,100)
        System . out . print (" Collection : ");
        stream (li). forEach (i-> System . out. print (" "+i));
    }

}
```

Il passo successivo, senza le lambda-expression, è illustrato nell' esempio seguente dove il programma utilizzando la classe Longstream permette di inserire i primi 1000 numeri Long all'interno di una lista. Le lambda-expression sono state rimpiazzate determinando però un codice più corposo:

```
package testStream;

import java.util.List;
import java.util.function.LongFunction;
import java.util.function.LongUnaryOperator;
import java.util.stream.Collectors;
import java.util.stream.LongStream;


public class Main {
    public static void main(String[] args) {

        List<? extends Object> asList = LongStream.iterate(1, new LongUnaryOperator() {
            @Override
            public long applyAsLong(long arg0) {
                return arg0 + 1;
            }
        }).limit(1000).mapToObj(new LongFunction<Object>() {

            @Override
            public Object apply(long arg0) {
                return arg0;
            }
        }).collect(Collectors.toList());
        System.out.println(asList);
    }
```

Utilizzando la libreria Streamsupport otteniamo lo stesso risultato l'unico accorgimento da usare è cambiare le import del progetto e usare la classe LongStreams:

```
package testStream;

import java.util.List;

import java8.util.function.LongFunction;
import java8.util.function.LongUnaryOperator;
import java8.util.stream.Collectors;
import java8.util.stream.LongStreams;

public class Main {

    public static void main(String[] args) {

        List<? extends Object> asList = LongStreams.iterate(1, new LongUnaryOperator() {

            @Override
            public long applyAsLong(long arg0) {
                return arg0 + 1;
            }
        }).limit(1000).mapToObj(new LongFunction<Object>() {

            @Override
            public Object apply(long arg0) {
                return arg0;
            }
        }).collect(Collectors.toList());
        System.out.println(asList);

    }
```

3.3 Retrolambda

3.3.1 Descrizione funzionalità

Analogamente a Retroweaver per l'esecuzione di codice Java 5 con i generici in Java 1.4, Retrolambda consente di eseguire codice Java 8 con le lambda-expression, method references e try-with-resources statements su Java 7, 6 o 5. Lo fa trasformando Java 8 bytecode compilato in modo che può essere eseguito su un runtime Java più vecchio. Dopo la trasformazione sono solo un gruppo di normali file .class, senza dipendenze runtime aggiuntivi. C'è anche un supporto limitato per il backport di deafult-methods e metodi statici su interfacce. Questa funzione è disabilitata per default.

Retrolambda supporta backporting a Java 7, Java 6 e Java 5 runtime. In più ci sono altri strumenti backport che possono portare Java 5 a Java 1.4.
Gli sviluppatori di Android possono utilizzare Retrolambda per sfruttare le caratteristiche di Java 8 su Android: vi è un plugin Gradle che rende facile tale processo.
Retrolambda non esegue il backport delle nuove Java 8 API, ma ci sono altri strumenti che svolgono questo compito:

- Streamsupport backport l'API java.util.stream
- ThreeTen backport l'API java.time [16]

3.3.2 JVM Invoke

JVM Invoke indica un set di istruzioni del compilatore utilizzato per gestire l'inizializzazione delle istruzioni, riconducibili ai metodi, del compilatore.
Nel set di istruzioni del compilatore Java8, così come per Java7 si fa uso delle istruzioni INVOKEVIRTUAL, INVOKESPECIAL ed INVOKEDYNAMIC.

INVOKEVIRTUAL è una istruzione bytecode che viene utilizzata per creare il riferimento ad un'istanza di un metodo di una classe.

public class INVOKEVIRTUAL
extends InvokeInstruction

INVOKESPECIAL è un'istruzione bytecode che viene utilizzata per creare il riferimento ad un'istanza di un metodo; è utilizzato in presenza di metodi legati a superclassi, a metodi privati e metodi d'inizializzazione (costruttori) [17]:

public class INVOKESPECIAL
extends InvokeInstruction

INVOKEDYNAMIC è un'istruzione bytecode che facilita l'implementazione dei dynamic languages (per la JVM) attraverso l'invocazione dinamica dei metodi. Questa istruzione è descritta nelle specifiche di Java SE 7 Edition della JVM [18].

La classe INVOKEDYNAMIC non è una istanza di InvokeInstruction, e si aspetta di ottenere la classe del metodo a runtime [19]:

public class INVOKEDYNAMIC
extends NameSignatureInstruction
implements ExceptionThrower, StackConsumer, StackProducer

3.3.3 Traduzione lambda-expression

Retrolambda permette di catturare le sezioni in cui, nel bytecode, viene invocato il riferimento a java.lang.invoke.LambdaMetafactory generando nuovo codice dinamicamente e traducendolo in chiamate diverse compatibili con il JDK7.

In Java, una lambda-expression può essere utilizzata per rimpiazzare l'utilizzo di una classe anonima permettendo di definire il corpo del run() dell'interfaccia Runnable:

```
1  package main;
2
3  public class Main {
4
5      public static void main(String[] args){
6          Runnable r = () -> System.out.println("hello world");
7          r.run();
8      }
9
10 }
11
```

Ad avvenuta compilazione, nel bytecode della classe Main l'istruzione INVOKEDYNAMIC viene utilizzata per gestire l'invocazione del metodo run() sull'oggetto r, che ancora non è associato ad un tipo:

```
studente@UniboCorsi ~/Workspace tesi/Test Lambda Expression/bin/main $ javap -c Main.class
Compiled from "Main.java"
public class main.Main {
  public main.Main();
    Code:
       0: aload_0
       1: invokespecial #8                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: invokedynamic #19,  0             // InvokeDynamic #0:run:()Ljava/lang/Runnable;
       5: astore_1
       6: aload_1
       7: invokeinterface #20,  1           // InterfaceMethod java/lang/Runnable.run:()V
      12: return
}
```

Se si guarda l'effettiva istruzione invokeDynamic si nota che non c'è alcun riferimento della funzione Lambda attuale (chiamato lambda $ 0). La risposta sta nella progettazione del metodo invokeDynamic. Il nome e la firma della lambda-expression sono memorizzati in una tabella separata del file.class nel quale vengono identificati con il parametro #1 con il riferimento: main/Main.lambda$:()V [20]:

```
  #52 = MethodHandle       #6:#46         // invokestatic java/lang/invoke/LambdaMetafactory.metafactory:(Ljava/lang/invoke/MethodHan
dles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodHandle;Ljava/lang/inv
oke/MethodType;)Ljava/lang/invoke/CallSite;
  #53 = MethodType         #6             //  ()V
  #54 = Methodref          #1.#55         // main/Main.lambda$0:()V
  #55 = NameAndType        #28:#6         // lambda$0:()V
  #56 = MethodHandle       #6:#54         // invokestatic main/Main.lambda$0:()V
  #57 = MethodType         #6             //  ()V
  #58 = Utf8               InnerClasses
```

3.3.4 Esempio di Retrolambda da linea di comando

Dopo aver scaricato l'ultima versione di Retrolambda dal sito ufficiale, si deve utilizzare JDK8 per compilare il codice sorgente.
Dalla console ci si posiziona nella cartella, dove è presente Retrolambda e lo si esegue sui file .class prodotti dal compilatore.

I file .class adesso dovrebbero girare su Java 7 o versioni più vecchie ancora:

```
studente@UniboCorsi ~/Workspace tesi $ java -Dretrolambda.inputDir=Test\ Lambda\ Expression/bin/ -Dretrolambda.classpath=Test\
\ Expression/bin -jar retrolambda-2.0.6.jar
Retrolambda 2.0.6
Bytecode version: 51 (Java 7)
Default methods:  false
Input directory:  Test Lambda Expression/bin
Output directory: Test Lambda Expression/bin
Classpath:        Test Lambda Expression/bin
Saving lambda class: main/Main$$Lambda$1
```

In questo modo il bytecode viene tradotto:

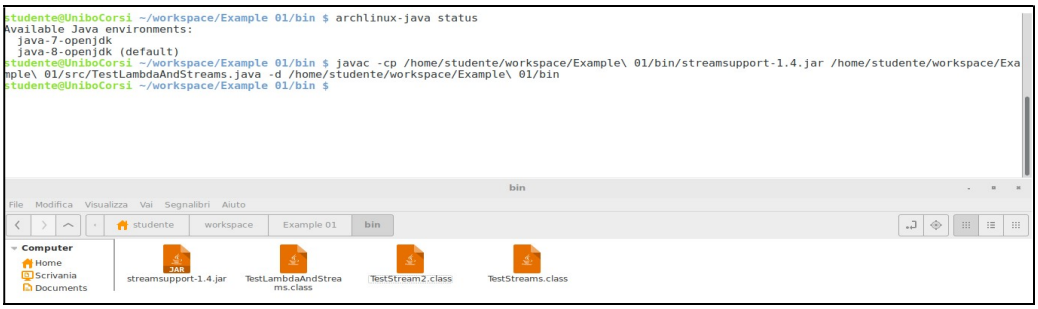
```
studente@UniboCorsi ~/Workspace tesi/Test Lambda Expression/bin/main $ dir
Main.class  Main$$Lambda$1.class
studente@UniboCorsi ~/Workspace tesi/Test Lambda Expression/bin/main $ javap -c Main
Warning: Binary file Main contains main.Main
Compiled from "Main.java"
public class main.Main {
  public main.Main();
    Code:
       0: aload_0
       1: invokespecial #14                 // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: invokestatic  #24                 // Method main/Main$$Lambda$1.lambdaFactory$:()Ljava/lang/Runnable;
       3: astore_1
       4: aload_1
       5: invokeinterface #29,  1           // InterfaceMethod java/lang/Runnable.run:()V
      10: return

  static void access$lambda$0();
    Code:
       0: invokestatic  #51                 // Method lambda$0:()V
       3: return
}
```

Il bytecode fa riferimento alla classe Main$$Lambda$1 che si trova all'interno della cartella dove vi era il .class originario.
In questo modo si effettua una traduzione del puntatore al metodo run() che in JDK8 avviene in automatico:

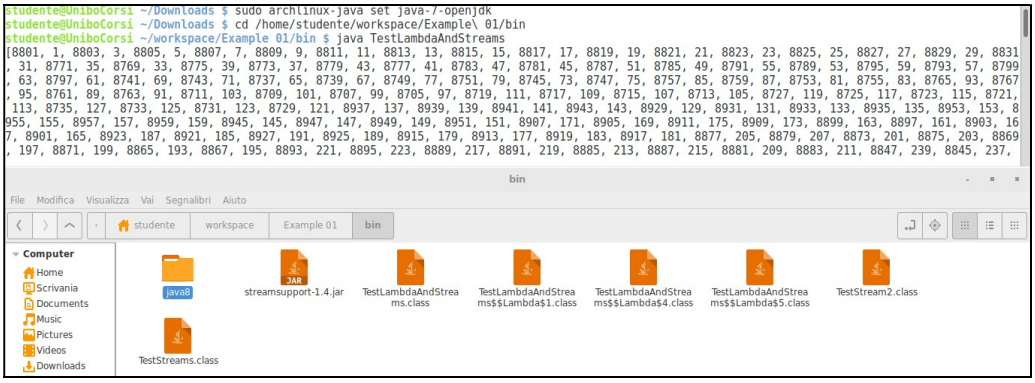
```
studente@UniboCorsi ~/Workspace tesi/Test Lambda Expression/bin/main $ javap -c Main\$\$Lambda\$1.cla
final class main.Main$$Lambda$1 implements java.lang.Runnable {
  public void run();
    Code:
       0: invokestatic  #17                 // Method main/Main.access$lambda$0:()V
       3: return

  static {};
    Code:
       0: new           #2                  // class main/Main$$Lambda$1
       3: dup
       4: invokespecial #21                 // Method "<init>":()V
       7: putstatic     #23                 // Field instance:Lmain/Main$$Lambda$1;
      10: return

  public static java.lang.Runnable lambdaFactory$();
    Code:
       0: getstatic     #23                 // Field instance:Lmain/Main$$Lambda$1;
       3: areturn
}
```

3.3.5 Retrolambda con Streamsupport

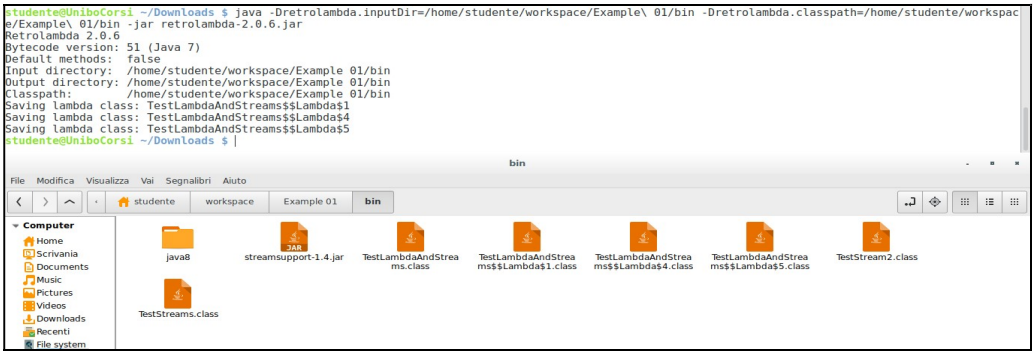
Integrare Streamsupport con Retrolambda non richiede procedimenti particolari se non quelli già discussi in precedenza. Qualora si volesse compilare il progetto da console però bisogna assicurarsi che si stia utilizzando la versione JDK 8:



Come prova basta compilare il sorgente contenente le lambda e gli stream con JDK 8 ed eseguirlo con con JDK7 per verificare che il compilatore fallisce:



Eseguendo invece Retrolambda sul sorgente permette invece di far girare la nostra applicazione con le lambda e gli stream su una JVM7:



3.4 Creazione di un’estensione del compilatore

Sulla piattaforma Android, il codice sorgente di Java è ancora compilato nel file .class. Dopo che i file .class vengono generati, lo strumento "dx" è utilizzato per convertirli in un unico file .dex o Dalvik Executable. Considerando che un file .class contiene una sola classe , un file .dex contiene più classi. E ' il file .dex che viene eseguito sul Dalvik VM [21] .

Una strategia per poter rendere il bytecode, presente all’interno dei file .class, compatibile con Android è di crearsi una propria estensione del compilatore Java in modo tale da eseguire una traduzione delle istruzioni di bytecode prodotte dal compilatore Java8 con quelle compatibili con un compilatore Java7.
Esistono già dei progetti che hanno svolto questo compito in precedenza.

Quando venne pubblicata la prima versione di Java con generici J2SE 5.0 vi era il problema di fornire al compilatore informazioni a livello di runtime sui tipi generici. Diverse soluzioni sono state studiate per risolvere questo problema. Da un lato, la JVM poteva essere riprogettata per rappresentare direttamente tipi generici,

eventualmente limitando le modifiche al class-loader. D'altra parte, i generici potevano essere reificati dal compilatore, in modo da creare automaticamente ulteriore codice in esecuzione sulla JVM esistente. I due approcci che hanno seguito queste direzioni sono NextGen ed EGO [22].

Al momento non esiste una versione di compilatore che permette di produrre un bytecode compatibile con Java 7 da un codice sorgente Java 8. Qualora si volesse sviluppare quindi un nuovo compilatore per Java 8, potrebbe utilizzato Polyglot. Esso è un compilatore front-end altamente estensibile per il linguaggio di programmazione Java. E' implementato come un framework di classe Java utilizzando modelli di progettazione per promuovere l'estensibilità. Polyglot è stato utilizzato sia per le estensioni del linguaggio Java maggiori e minori; in genere il costo per sviluppare un'estensione personalizzata scala bene con il grado in cui essa modifica Java.

Polyglot supporta Java 1.4, Java 5 e Java 7, tra cui funzioni come farmaci generici e le annotazioni. Il supporto per Java 5 e Java 7 è fornito come estensioni di Polyglot è inoltre possibile generare versioni di Java diverse come output [23].

4 PROGETTO E IMPLEMENTAZIONE

4.1 Integrazione degli strumenti di backport

In seguito ai diversi tentavi, svolti in laboratorio, ho studiato il funzionamento degli strumenti disponibili a oggi per effettuare il backport ho iniziato a capire l'importanza di Gradle. Avere uno strumento che semplifica l'integrazione di diverse funzionalità all'interno di un progetto, in maniera modulare, è di fondamentale importanza.

Nell'esempio di Protelis, essendo un'applicazione di medie dimensioni, circa cinquanta classi, ho deciso di raggiungere l'obiettivo finale seguendo il processo di revisione manuale del codice rispettando una certa solidità nelle fasi per la riprogettazione.

Dopo aver stabilito che effettivamente Streamsupport e Retrolambda funzionassero separatamente ho quindi cercato di integrarli insieme all'interno di uno stesso progetto, tramite l'utilizzo di Gradle.

4.1.1 Retrolambda con Gradle

Per integrare Retrolambda in uno script Gradle bisogna trascrivere alcune informazioni per definire quale compito e come lo dovrà eseguire:

1. Download jdk8.

2. Add the following to your build.gradle

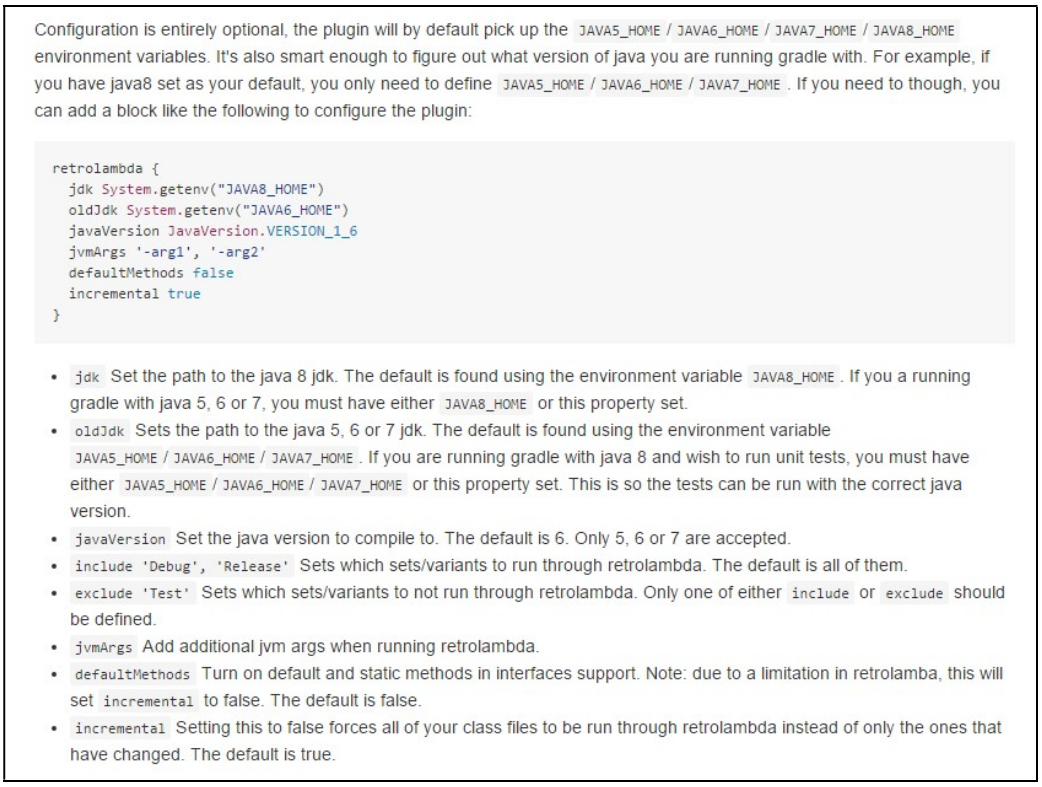
```
buildscript {
  repositories {
     mavenCentral()
  }

  dependencies {
     classpath 'me.tatarka:gradle-retrolambda:3.2.5'
  }
}

// Required because retrolambda is on maven central
repositories {
  mavenCentral()
}

apply plugin: 'com.android.application' //or apply plugin: 'java'
apply plugin: 'me.tatarka.retrolambda'
```

Note: If you are using the `2.0.0-alpha/beta` android gradle plugins, you should use `3.3.0-beta4` since it better supports instant run.

alternatively, you can use the new plugin syntax for gradle `2.1+`

In generale per configurare Retrolambda, da Gradle, si possono inizializzare alcuni flag, l'esempio in fattispecie indica il PATH del JDK con cui deve essere compilato il progetto, con il quale devono essere eseguiti i test e la versione di Java alla quale si vuole tradurre il bytecode.

Per eseguire lo script gradle, oltre che dall'IDE da cui si lavora (in caso di Eclipse), basta lanciare una semplice gradle build oppure una ./gradlew dalla cartella root del progetto.



Configuration is entirely optional, the plugin will by default pick up the `JAVA5_HOME` / `JAVA6_HOME` / `JAVA7_HOME` / `JAVA8_HOME` environment variables. It's also smart enough to figure out what version of java you are running gradle with. For example, if you have java8 set as your default, you only need to define `JAVA5_HOME` / `JAVA6_HOME` / `JAVA7_HOME`. If you need to though, you can add a block like the following to configure the plugin:

```
retrolambda {
  jdk System.getenv("JAVA8_HOME")
  oldJdk System.getenv("JAVA6_HOME")
  javaVersion JavaVersion.VERSION_1_6
  jvmArgs '-arg1', '-arg2'
  defaultMethods false
  incremental true
}
```

- `jdk` Set the path to the java 8 jdk. The default is found using the environment variable `JAVA8_HOME`. If you a running gradle with java 5, 6 or 7, you must have either `JAVA8_HOME` or this property set.
- `oldJdk` Sets the path to the java 5, 6 or 7 jdk. The default is found using the environment variable `JAVA5_HOME` / `JAVA6_HOME` / `JAVA7_HOME`. If you are running gradle with java 8 and wish to run unit tests, you must have either `JAVA5_HOME` / `JAVA6_HOME` / `JAVA7_HOME` or this property set. This is so the tests can be run with the correct java version.
- `javaVersion` Set the java version to compile to. The default is 6. Only 5, 6 or 7 are accepted.
- `include 'Debug', 'Release'` Sets which sets/variants to run through retrolambda. The default is all of them.
- `exclude 'Test'` Sets which sets/variants to not run through retrolambda. Only one of either `include` or `exclude` should be defined.
- `jvmArgs` Add additional jvm args when running retrolambda.
- `defaultMethods` Turn on default and static methods in interfaces support. Note: due to a limitation in retrolamba, this will set `incremental` to false. The default is false.
- `incremental` Setting this to false forces all of your class files to be run through retrolambda instead of only the ones that have changed. The default is true.

4.1.2 Retrolambda con Streamsupport e Gradle

Mentre Retrolambda è un plugin, in altre parole un componente modulare a se stante, per quanto riguarda l'utilizzo di StreamSupport tramite Gradle bisogna come prima cosa indicare l'URL specifico relativo al repository MavenCentral. Per compilare ed eseguire Retrolambda tramite Eclipse utilizzando Gradle è possibile definire i vari task e la loro successione configurando il file build gradle.

Gradle come primo compito si occupa di scaricare le dipendenze delle librerie, utilizzate nell'applicazione, dal repository MavenCentral, in seguito avvia la compilazione esegue i default task e applica il plugin per gradle di Retrolambda [24].

Per compilare ed eseguire il progetto Gradle, da linea di comando, bisogna indicare al compilatore dove sono le dipendenze scaricate da gradle, esse vengono scaricate e mantenute in una cache temporanea all'interno del sistema operativo. La compilazione da console può essere utile a livello didattico però, in pratica, è piuttosto scomoda:

```
java -cp ~/.gradle/caches/modules-2/files-
2.1/net.sourceforge.streamsupport/streamsupport/1.4/61b40927
2deff2f224c3c0c37ae2d9007817696/streamsupport-
1.4.jar:build/libs/Example\ 01.jar Main
```

4.2 Backport Protelis

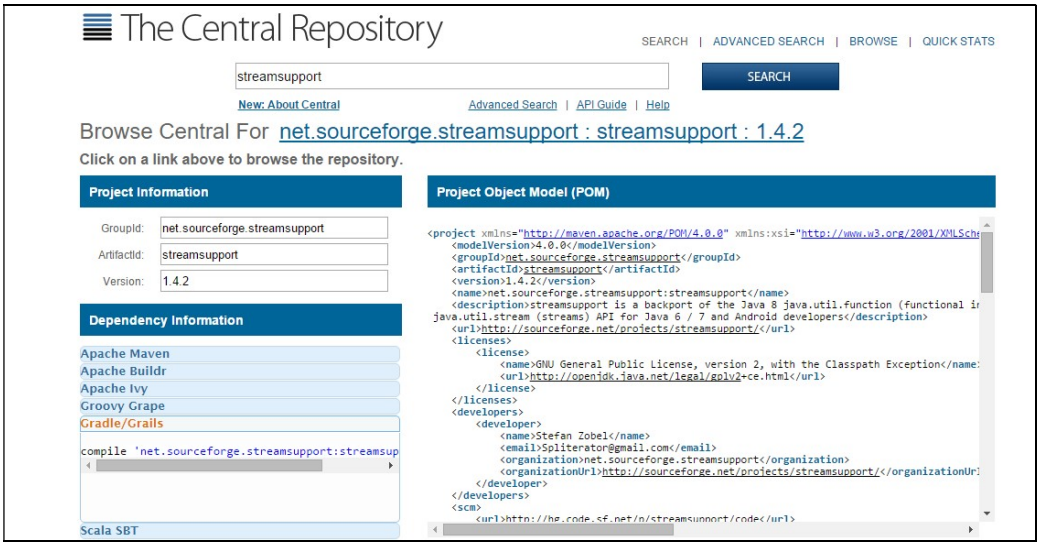
Protelis per poter assicurare una struttura sicura e resiliente ha il nucleo costituito da un modello teorico di programmazione aggregata molto simile alla funzionalità delle lambda [10].

Protelis sfrutta tutti gli strumenti messi a disposizione con Java 8. E' stato scritto facendo abbondante uso degli stream della libreria java.util.stream, delle lambda-expression e dei metodi statici nelle interfacce.

Il metodo utilizzato per eseguire il backporting di tale software utilizza un procedimento incrementale in modo da poter procedere per fasi successive nelle quali l'output di una fase costituisce l'input della fase successiva. E' importante quindi avere chiaro dal principio com'è strutturato il software che si andrà a manipolare e come suddividere il flusso di lavoro. Per mantenere una visione d'insieme ben definita e, per recuperare il progetto in caso di errori di progettazione, si può utilizzare lo strumento di subversioning Git.

4.2.1 Integrazione di Streamsupport

Nell'effettuare il backport di Protelis, bisogna per prima cosa integrare la libreria Streamsupport attraverso l'utilizzo di Gradle [15].
Tale libreria, è stata pubblicata, dai suoi sviluppatori anche nel repository Maven Central per cui è possibile procedere con l'integrazione di essa nel progetto.
Una volta cercata la libreria all'interno del sito, basta copiare il link, nella sezione Gradle/Grails, e inserirlo all'interno delle dependecies di file build.gradle:



Per includere la libreria di Streamsupport all'interno del progetto si agisce quindi unicamente sul file build.Gradle presente in ciascun progetto Gradle:

```
10
11 dependencies {
12     compile 'net.sourceforge.streamsupport:streamsupport-flow:1.4'
13   }
14
15
```

4.2.2 Tabella delle conversioni

In Protelis, come già detto, si fa uso sia di Stream, di Lambda-express e di altre librerie introdotte con Java 8 come Java.Util.Optional e Java.Util.Function.

Il primo compito della procedura di backport in Protelis consiste nella sostituzione quindi delle import di queste librerie non supportate da Java 7 con la nuova libreria Streamsupport appena importata all'interno del progetto e alla sostituzione dei metodi all'interno delle diverse classi.

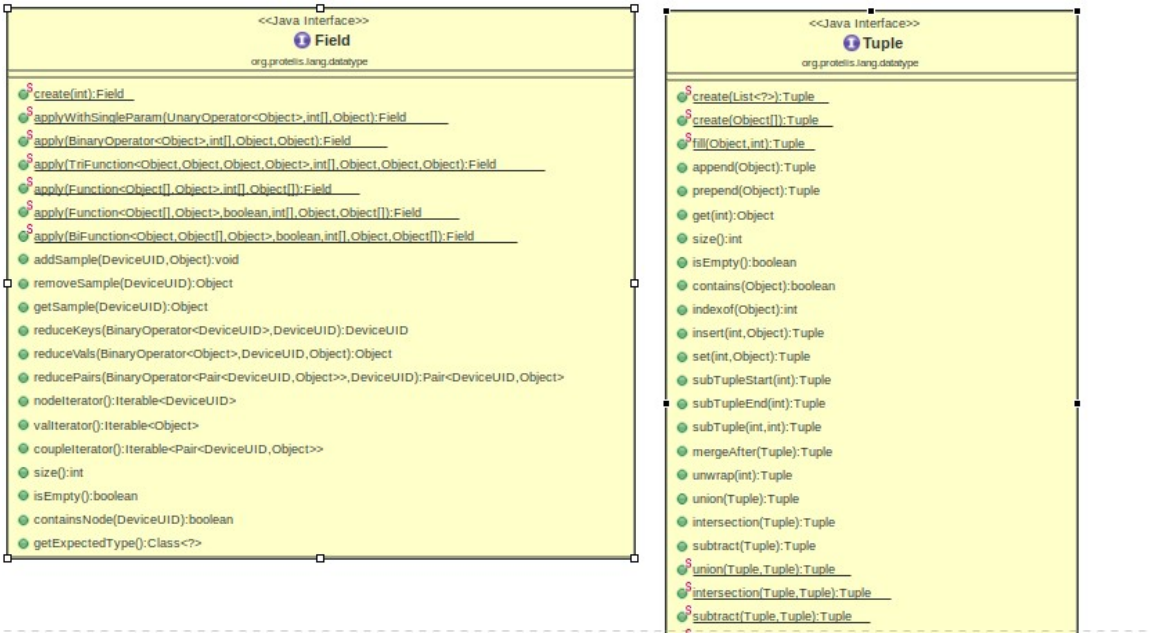
Ecco una tabella riassuntiva che elenca i principali cambiamenti da apportare all'applicazione:

Librerie di Java 8	**Libreria di streamsupport**	**Descrizione**
Collection.stream()	static java8.util.stream.Streamsupport.stream	Con Streamsupport per creare uno stream da una lista basta invocare il metodo statico della libreria su una collezione.
java.util.stream.*parallel*	java8.util.stream.BaseStream.*parallel*	Con Streamsupport è anche possibile sfruttura la potenza di un'architettura multicore in quanto è stata riprodotta l'istruzione parallel che ritorna un parallel stream.
java.util.stream	Java8.util.stream.RefStream	All'interno di RefStream vi sono tutti i metodi statici presenti

		nell'interfaccia Stream di Java 8.
java.util.function	java8.util.function	Tutti i metodi presenti in java.util.function sono stati riscritti all'interno di Streamsupport.
java.util.Arrays	java8.util.J8Arrays	Per creare uno Stream da un'array in Java 7 bisogna invocare il metodo statico della omonima libreria di Streamsupport passandogli come parametro l'array.
java.util.Optional	java8.util.Optional	Tutti i metodi presenti in java.util.Optional sono stati riscritti all'interno di streamsupport
Java.util.Map	java8.util.Maps	Per iterare gli elementi di una mappa, per mappare ciascuna coppia chiave-valore di una mappa attraverso una funzione utilizzare il metodo statico presente nella omonima libreria di streamsupport

4.2.3 Conversione dei metodi statici nelle interfacce

In Protelis, in particolare nelle interfacce Field e Tuple si fa uso di metodi statici che sfortunatamente non sono supportati in Java 7. E' stato scelto questo approccio di programmazione per evitare l'utilizzo di classi rindonanti:



Tali metodi, come le lambda-expression, si riconducono alla nuova istruzione del compilatore invokeDynamic e quindi costituiscono un problema per cui bisogna trovare una soluzione.
Retrolambda ha solo un supporto parziale per questo problema, infatti, dopo innumerevoli tentativi ho constatato che per Protelis non riesce a tradurre correttamente il bytecode. La soluzione è stata quella di utilizzare una classe Factory che implementa i metodi statici presenti e tale classe viene quindi invocata al posto delle interfacce.

Nei diagrammi della classi è bene notare come devono essere re implementati i metodi statici (notare la s):

4.2.4 Integrazione di Retrolambda

Dopo aver rimpiazzato le API presenti da Java 8 e i metodi statici nelle interfacce rimane il problema di convertire le lambda-expression.

Esse potrebbero essere rimpiazzate dalla classi anonime però tale procedimento incrementerebbe notevolmente, in termini di tempo, il processo di backporting. Siccome Protelis dispone già di un'architettura Gradle al suo interno, utilizziamo quindi il plugin di Retrolambda per convertire il bytecode del sorgente in modo da poter lasciare le lambda-expression all'interno dell'applicazione.

Inserendo il classpath di Retrolambda all'interno delle dipendenze del buildscript e creando il plugin ad esso associato otteniamo il risultato atteso:

```
apply plugin: 'java'
apply from: 'eclipse.gradle'
apply from: 'codequality.gradle'
apply plugin: 'project-report'
apply from: 'maven.gradle'
apply plugin: 'build-dashboard'
apply from: 'signing.gradle'

buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
       classpath 'org.danilopianini:smartrrr:[0, 1['
        classpath 'me.tatarka:gradle-retrolambda:3.2.4'
    }
}
apply plugin: 'org.danilopianini.smartrrr'
apply plugin: 'me.tatarka.retrolambda'

retrolambda {
  jdk System.getenv("JAVA8_HOME")
  oldJdk System.getenv("JAVA7_HOME")
  javaVersion JavaVersion.VERSION_1_7
  defaultMethods true
  incremental false
}
```

4.2.5 Verifica dell'effettivo funzionamento di Java 7

Per verificare l'effettivo funzionamento del progetto di cui è stato fatto il backport a java 7 va eseguito, da console, il comando ./gradlew :

```
studente@UniboCorsi ~/protelis feature-backport/Protelis $ ./gradlew
Download https://repo1.maven.org/maven2/org/danilopianini/javalib/5.0.0/javalib-
5.0.0.pom
Download https://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.7.18/slf4j-api-1.7
.18.pom
Download https://repo1.maven.org/maven2/org/slf4j/slf4j-parent/1.7.18/slf4j-pare
```

Avendo aggiunto il task fatJar nel nostro file di configurazione build.gradle possiamo anche lanciare l'applicazione da linea comando con il classico comando java –jar fileJar.

```
143 task fatJar(type: Jar) {
144     baseName = project.name + '-redist'
145     from(configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }) {
146         exclude "META-INF/*.SF"
147         exclude "META-INF/*.DSA"
148         exclude "META-INF/*.RSA"
149     }
150     with jar
151 }
```

CONCLUSIONI

Il backport di un progetto è un processo informatico attraverso il quale è possibile eseguire un software su un sistema predisposto con una versione di compilatore inferiore a quella richiesta per il software in questione.

Idealmente, la modalità di approccio allo sviluppo di un processo di backport troverebbe la massima ottimizzazione seguendo una logica di automazione, questo però si traduce con un impiego di risorse uomo/macchina maggiore.

Occorre valutare le dimensioni del problema e in base a ciò scegliere la strada più adatta. Nel caso in cui si debba effettuare il backport di un'applicazione molto grossa, o di più molteplici piccole applicazioni allora conviene indirizzarsi sullo sviluppo di un sistema di traduzione automatico di codice o bytecode.

Nel caso in cui si debba effettuare il backport di un singolo progetto, di piccole o medie dimensioni (es. Protelis) allora per ottimizzare i tempi di lavoro, conviene procedere manualmente integrando Retrolambda e Streamsupport all'interno del progetto.

Questo approccio quindi evita che il processo di backporting di un'applicazione si ripercuota, quando non necessario, sul rapporto dei costi uomo/macchina.

BIBLIOGRAFIA

1. https://it.wikipedia.org/wiki/Android. Ultimo accesso 27/02/2016
2. https://it.wikipedia.org/wiki/Backport. Ultimo accesso 27/02/2016
3. https://it.wikipedia.org/wiki/Java_%28linguaggio_di_programmazione%29#Descrizione. Ultimo accesso 27/02/2016
4. https://www.java.com/it/download/faq/java8.xml. Ultimo accesso 27/02/2016
5. https://it.wikipedia.org/wiki/Gradle. Ultimo accesso 28/02/2016
6. https://appissue.wordpress.com/2015/06/16/gradle-da-dove-iniziare-principi-fondamentali-build-language-reference-2015. Ultimo accesso 28/02/2016
7. https://docs.gradle.org/current/dsl/org.gradle.api.Task.html. Ultimo accesso 28/02/2016
8. https://docs.gradle.org/current/userguide/tutorial_using_tasks.html. Ultimo accesso 28/02/2016
9. https://en.wikipedia.org/wiki/Artifact_(software_development). Ultimo accesso 28/02/2016
10. http://protelis.github.io/. Ultimo accesso 29/02/2016
11. https://www.java.com/it/download/faq/java8.xml. Ultimo accesso 27/02/2016
12. http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html. Ultimo accesso 27/02/2016
13. M. Viroli. Programmazione ad oggetti. Appunti di lezione AA 2015-2016
14. http://wiki.apidesign.org/wiki/InvokeDynamic. Ultimo accesso: 28/02/2016
15. https://sourceforge.net/projects/streamsupport/. Ultimo accesso 27/02/2016
16. https://github.com/orfjackal/retrolambda. Ultimo accesso 27/02/2016

17. https://commons.apache.org/proper/commons-bcel/apidocs/org/apache/bcel/generic/InvokeInstruction.html. Ultimo accesso 28/02/2016
18. http://www.javaworld.com/article/2860079/scripting-jvm-languages/invokeDynamic-101.html. Ultimo accesso 28/02/2016
19. https://commons.apache.org/proper/commons-bcel/apidocs/org/apache/bcel/generic/INVOKEDYNAMIC.html. Ultimo accesso 28/02/2016
20. http://blog.takipi.com/compiling-lambda-expressions-scala-vs-java-8/. Ultimo accesso 28/02/2016
21. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf. Ultimo accesso 29/02/2016
22. http://www.sciencedirect.com/science/article/pii/S016764230800066X. Ultimo accesso 29/02/2016
23. http://www.cs.cornell.edu/Projects/polyglot/. Ultimo accesso 27/02/2016
24. https://github.com/evant/gradle-retrolambda. Ultimo accesso 27/02/2016