Università degli Studi di Verona

CORSO DI LAUREA IN INFORMATICA

Programmazione Java

Federico Brutti
federico.brutti@studenti.univr.it

Inserire citazione inerente alla materia

Contents

Chapter 1

Introduzione

1.1 Programmazione orientata agli oggetti

Il corso di programmazione II ha lo scopo di fornire un modus operandi differente; passeremo infatti dalla programmazione imperativa ad una **orientata agli oggetti**.

Lo scopo e l'utilità della OOP è quello di gestire appropriatamente un programma con molte linee di codice. Rende il codice estremamente modulare, riutilizzabile e di conseguenza risulta più facile mantenerlo. Per il modo in cui sono trattati gli oggetti, inoltre, il codice è reso particolarmente sicuro.

Familiarizziamo ora con i due concetti base della OOP: **classi** e **oggetti**. Le prime definiscono una struttura dati, alla quale è possibile associare dati, detti **attributi**, e funzioni, chiamate **metodi**. Dalle classi, che in parole povere fungono da tipo di dato, è possibile ottenere gli oggetti, delle loro istanze, con stessi dati e metodi. Infatti, il processo di creazione di un oggetto è detto **istanziazione**.

Il workflow della OOP si basa sulle interazioni fra gli oggetti; queste avvengono grazie alle loro **interfacce**, set di messaggi che l'oggetto può ricevere, mappate ad un metodo nello stesso. Se si riceve un messaggio al di fuori dell'interfaccia, è detto illegale e viene bloccato. Sicuro, no? Ora, l'idea generale per una corretta programmazione a oggetti sono:

1. Identificare i componenti.
2. Definire l'interfaccia dei componenti.
3. Definire le modalità con cui le interfacce consentono l'interazione fra oggetti.
4. Minimizzare le relazioni fra i componenti.

Uno strumento utile per la documentazione delle classi è l'**Unified Modeling Language**, consente di definire graficamente e quindi documentare un sistema orientato agli oggetti.

1.2 Il linguaggio Java

Il linguaggio utilizzato nel corso sarà Java, nato nel '95 e proprietà di Oracle, è diventato lo standard nel mercato del lavoro per il software development. La sua caratteristica principale è la portabilità, ovvero è possibile eseguire programmi su qualunque architettura grazie alla **Java Virtual Machine**, la quale funge da interprete al codice compilato.

Più precisamente, dopo essere compilato, verrà creato in output un file con estensione ".class" contenente il **bytecode**. Questo codice è ciò che viene effettivamente interpretato in runtime da una parte della JVM, il compiler **Just In Time**, utile anche per ulteriori ottimizzazioni. In soldoni, a patto che la macchina abbia installata la JVM, sarà possibile eseguire i files compilati. Altre caratteristiche di Java sono:

- Linguaggio fortemente tipizzato, ovvero è possibile dichiarare variabili di un determinato tipo di dato, le quali non lo possono cambiare una volta aggiunte al codice.
- Non ha manipolazioni esplicite di puntatori grazie alla filosofia dell'incapsulamento, rendendo meno probabili errori riguardanti la memoria.
- Controlla il runtime, rendendo impossibile avere array overflow.
- Il **Garbage collector** controlla eventuali leaks di memoria per tapparle.
- È possibile usare eccezioni per controllare gli errori.
- Linguaggio fortemente dinamico, poiché fa loading e linking in runtime. Inoltre, usa dimensioni di array dinamiche.

Dove è possibile installare sulla macchina solo la JVM, tipico se vuoi giocare a Minecraft, per scrivere programmi in Java è necessario usare il **Java Development Kit**, compreso di debugger, compiler, disassembler, ed un applicativo per la documentazione.

Per l'esecuzione dei programmi abbiamo poi il **Java Runtime Environment**, avente con sé librerie di classe, il compiler JIT precedentemente menzionato, la JVM e il Java application launcher.

Basta cazzate, la struttura generale di un programma Java è la seguente:

```
1  // Per compilare: javac HelloWorld.java
2  // Per eseguire: java HelloWorld
```

```
3
4   // Il nome qui deve coincidere con quello del file.
5   public class HelloWorld {
6     // Entry point del programma, funge da metodo.
7     public static void main (string[] args) {
8       // Blocco di codice
9     }
10  }
```

Ho menzionato prima come il compiler restituisca in output un file di estensione .class, infatti ogni file è visto come classe con i relativi oggetti. Il caricamento di queste classi è basato sul **classpath**, la lista di locazioni dove queste possono essere prese. Se il compiler non trova la classe, lancerà un'eccezione ed il programma non verrà eseguito.

Ci sono più modi per indicare un classpath al compilatore; indicarla singolarmente oppure contenere tutte le classi in un barattolo, un file con estensione ".jar". Questi sono fondamentalmente degli archivi compressi in cui è salvato il bytecode e altre meta-informazioni. Quindi:

- Per eseguire da cartelle diverse: $ java -cp ./nomeCartella nomeEseguibile
- Per creare un file .jar: $ jar cvf nomeJar.jar classOne.class classTwo.class ...
- Per eseguire un programma con file .jar: $ java -cp nomeJar.jar nomeEseguibile

Nella creazione di un file .jar è possibile aggiungere anche una direttiva su quali classi eseguire di preciso e quali ignorare. Ciò si fa con un file nominato **manifest.txt**.

```
1   // manifest.txt
2   Main-Class: nomeClasse
3
4   // Compilazione
5   $ jar cvfm nomeJar.jar manifest.txt classOne.class
```

Notare i termini **cvf** e **cvfm**, significanti rispettivamente create verbose file e create verbose file manifest.

1.3 Componenti del linguaggio

Si suppone che ti abbia frequentato il precedente corso di programmazione, tenuto con C. Conoscerlo faciliterà enormemente le cose, poiché i tipi di Java sono dichiarati nel medesimo modo. Manteniamo **int**, **double**, **char**, aggiungendo **boolean**, autoesplicativo. Non sarà primitivo, ma hanno aggiunto il tipo stringa, reso più sicuro con l'incapsulamento. Si possono eseguire concatenazioni con l'operatore "+". Non cambia nemmeno il type casting, se per qualche motivo dovesse servire, e va in ambo le direzioni. Le vere novità sono le classi e gli oggetti, comuni a tutti i linguaggi OOP:

```
1  // Dichiarazione di una classe
2
3  // Dichiarazione di un oggetto
```

– Oggetti Istanze di classi accessibili solamente tramite messaggi. Hanno dei campi, sottoinsiemi di locazioni di memoria, interagibili coi metodi. Java ha puntatori ma non ne esiste l'aritmetica.

Referenza dell'oggetto contiene il nome di una classe ed un indirizzo di memoria. Non usa memoria, tiene solo l'indirizzo.

class Vehicle // Rivedere la definizione di classe, potrei aver scritto na minchiata string Manifacturer; setFrameNumber(int); ; Vehicle MyCar; // Contiene Vehicle@null MyCar = new Vehicle(); // MyCar ora è un oggetto di classe Vehicle. Istanziazione. Il compiler vedrà Vehicle@memAddress MyCar.setFrameNumber(2); // Messaggio legale inviato alla zona di memoria memAddress.

– Ereditarietà; rivedere in autonomia (?) Puoi aggiungere altri dati e metodi dichiarando prima un oggetto di una classe, per poi assegnarli un new oggettoDiverso(); Vehicle myCar myCar = new Car(); // Un oggetto Car eredita tutti i campi e metodi degli oggetti vehicle

Puoi usare (objname == null) per controllare se qualche puntatore va a null per qualche ragione. Utile, dai. Come è anche possibile mettere un oggetto a null. Aiuta il garbage collector a capire quale memoria deallocare.

— Sintassi base — Costrutti di base - Le variabili si dichiarano come in C. Boom, risparmiato 30min di spiegazione. - Gli oggetti vanno inizializzati per evitare NullPointerException... ma dai? - Le costanti si dichiarano con "final static n = 1;"

Chapter 2

Example 2

2.1 Sec1

2.2 Sec2

2.2.1 SubS1