APPUNTI SU JAVA

Angelo Di Iorio
Università di Bologna

In questo laboratorio

- Enfasi su esercitazioni pratiche: partecipate (partecipate, partecipate) con un laptop
- Guardiamo alcune strutture dati implementate in Java ma ma soprattutto usiamo Java per re-implementare alcune strutture dati e algoritmi
- Useremo Eclipse ma non è obbligatorio

Eclipse

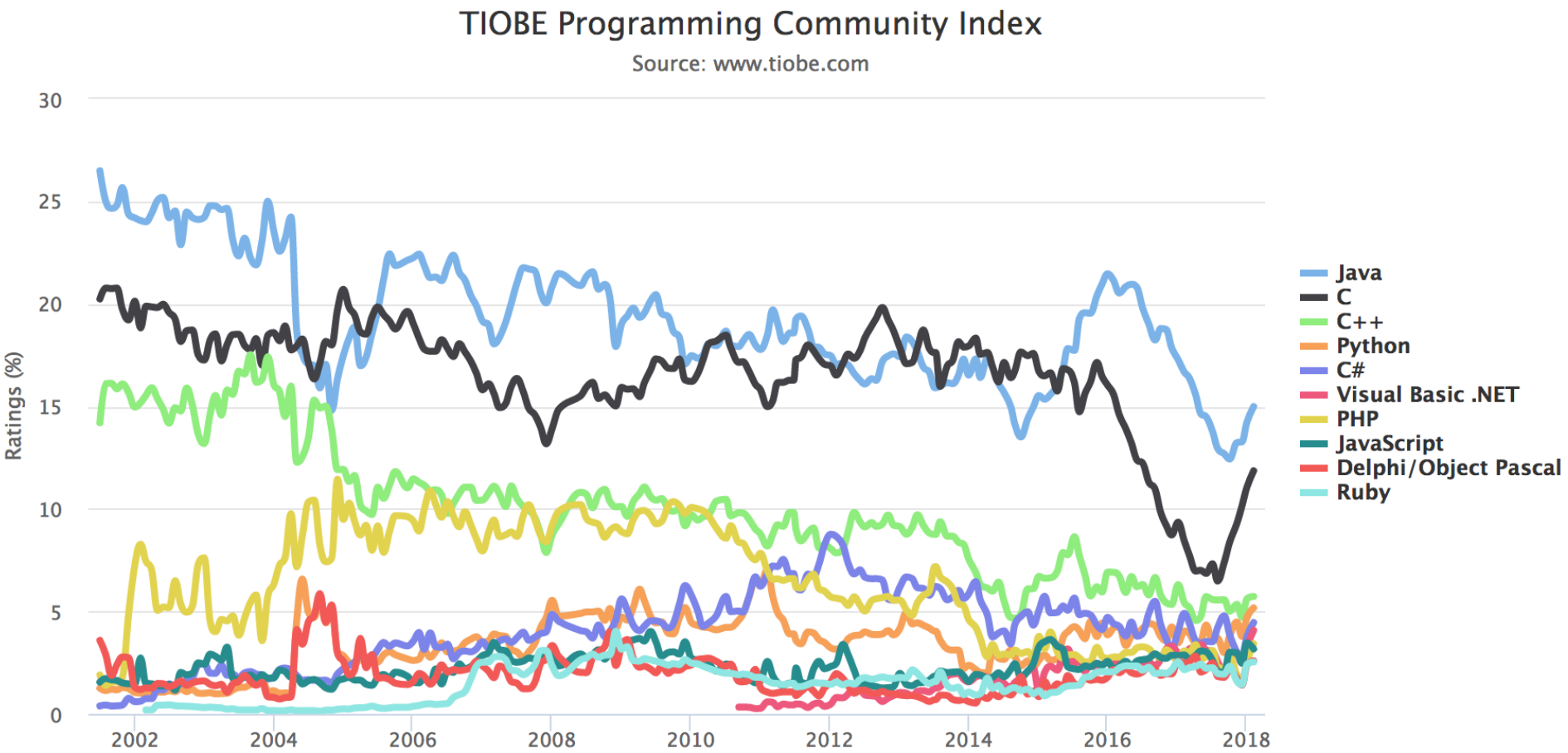
- Estratto da Wikipedia:
- *Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system.*
- *It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy and Scheme.*
- *Released under the terms of the Eclipse Public License, Eclipse is free and open source.*

- IDE
  - http://www.eclipse.org/ide/
- Archivio con tutte le release:
  - http://archive.eclipse.org/eclipse/downloads/

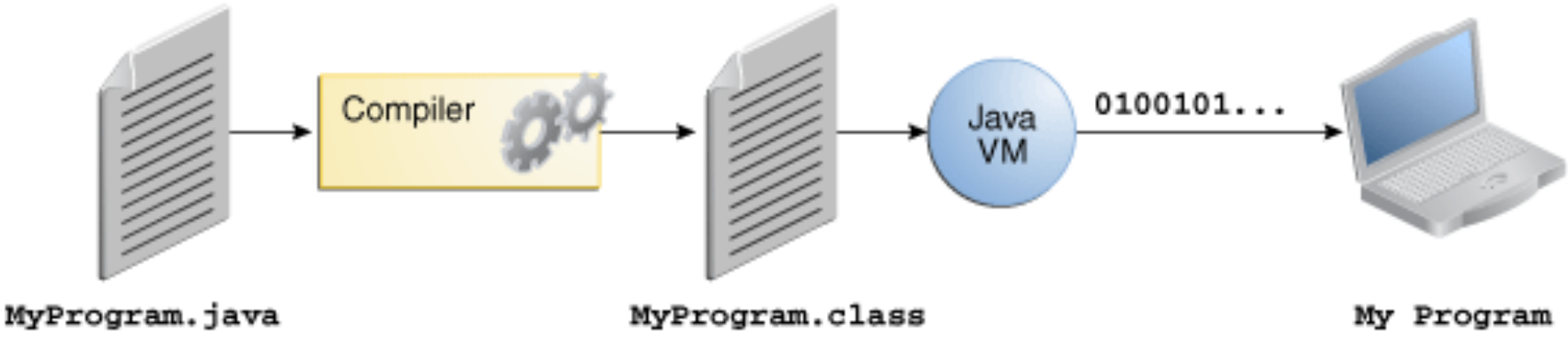
Java: un po' di storia

- Java è un linguaggio di programmazione general-purpose, object-oriented, multi-thread e platform-independent
- Ideato da James Gosling nel 1995
- Sintassi derivata dal C e C++, ma con meno funzionalità e controllo "a basso livello"
- Originariamente linguaggio proprietario (di Sun), rilasciato nel 2007 con licenza GPL
- Ultima release: Java 9 (21 settembre 2017)
- Anche Java 8 è largamente usato e supportato for-free da Oracle (che ha acquisito Sun)
- E' ancora il linguaggio di programmazione più usato

Per curiosità: TIOBE

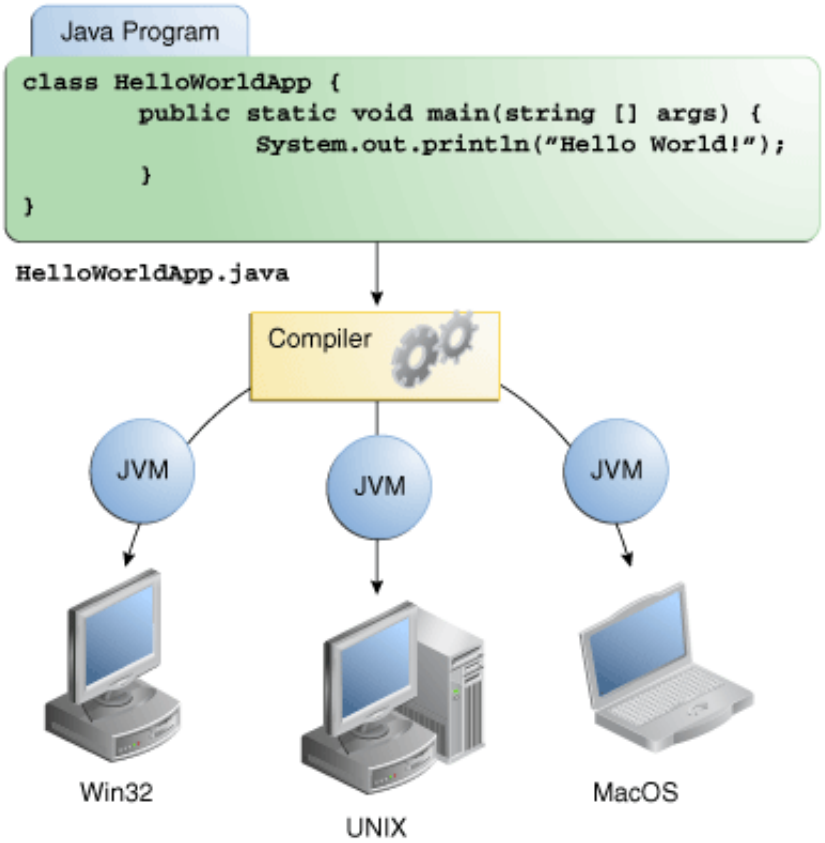


Java Development Process



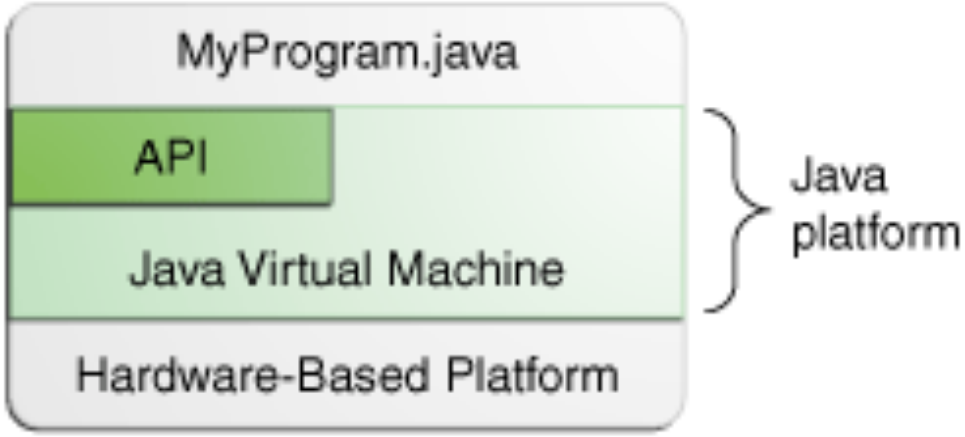
http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java Virtual Machine



http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

The Java Platform



http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Operatori, costrutti di base e tipi nativi

- Non guardiamo in dettaglio gli operatori e i comandi principali di Java, la cui sintassi è simile al C:
  - operatori aritmetici, booleani, di contronto, etc.
  - variabili e assegnamenti
  - comandi `if`, `else`, `while`, `for`, etc.

- Tipi primitivi:
  - *int*: numero intero (a 32 bit)
  - *byte*, *short*, *long*: numero intero di diversa lunghezza (8, 16 o 64 bit)
  - *boolean*: valori true/false
  - *char*: carattere
  - *float*, *double*: numeri decimali di diversa precisione

Tipi in Java

- Java è un linguaggio **fortemente tipato** (**strongly typed**): ogni espressione ha un tipo, che il compilatore usa per controllare la correttezza delle operazioni eseguite
  - molti errori sui tipi sono “anticipati” nell’IDE

- I tipi si distinguono in:
  - **tipi primitivi**
  - **tipi riferimento**: riferimento ad un oggetto, che può essere acceduto tramite appunto un riferimento che lo identifica in modo univoco

- **Garbage Collector**: la deallocazione di memoria in Java non è gestita dal programmatore in Java ma se ne occupa direttamente il supporto run-time che libera la memoria non utilizzata

Esercizio 0

- Scrivere un programma Java che stampa a video il messaggio “Hello World!”.

Object-Oriented Programming

- Java è nativamente object-oriented
- La programmazione orientata agli oggetti (OOP) è un paradigma di programmazione pensato per rendere più semplice lo sviluppo, l'evoluzione, la manutenzione e il testing del software
- Un paradigma diverso da quello imperativo (occorre ragionare un modo diverso!)
- Consente di scrivere le stesse applicazioni ma in modo diverso
- Fondamentale per scrivere applicazioni grandi ma utile anche per applicazioni piccole
- Favorisce il riuso

Object-Oriented Programming

- Un programma è visto come un **insieme di oggetti** che:
  - contengono **valori** e le **funzioni** che possono modificare tali valori
  - **interagiscono** tra loro

- Tre concetti fondamentali:
  - Incapsulamento
  - Ereditarietà
  - Polimorfismo

Terminologia: classi e oggetti

- Le **classi** sono **tipi di dato** definiti dal programmatore
- Una classe è composta da:
  - **attributi (o campi)**: variabili o costanti che definiscono le caratteristiche degli oggetti di quel tipo
  - **metodi**: procedure che operano sugli attributi

- Un **oggetto** è un'**istanza** di una **classe**

- Le classi consentono di:
  - definire il comportamento del dato
  - nascondere l'implementazione (***incapsulamento***)

Incapsulamento

- Un oggetto contiene una parte *pubblica* e una parte *privata* (anche una parte *protected*, ci torneremo a breve)
  - la parte pubblica è visibile da chi usa l'oggetto
  - la parte privata no
- La parte privata è incapsulata (*information hiding*)
- Si realizza così una netta separazione tra **interfaccia** e **implementazione**
- Se si modifica la parte privata lasciando inalterata quella pubblica si può continuare ad interagire con la parte pubblica
  - L'interfaccia dell'oggetto non cambia

Visibilità

- I metodi (e in generali le classi e le variabili di istanza) possono avere diversi livelli di accesso, definiti con apposite keyword prima del nome del metodo:
  - **`public`**: il metodo può essere invocato da tutte le altre classi
  - **`protected`**: il metodo può essere invocato da:
    - Sottoclassi della classe corrente (ereditarietà)
    - Classi appartenenti allo stesso package (modularità, ci torneremo a breve)
  - **`private`**: il metodo può essere invocato solo da altri metodi della stessa classe; né sottoclassi né altri classi possono accedervi

Costruttore

- Il costruttore è un particolare metodo di una classe che viene invocato al momento della creazione di un oggetto e permette di inizializzare lo stato interno dell'oggetto
- In Java il costruttore deve avere lo stesso nome della classe e non bisogna specificare il tipo di ritorno (ritorna un riferimento all'oggetto appena creato)
- Possono esistere più costruttori con lo stesso nome ma con un diverso numero di parametri
- Java supporta anche un metodo distruttore (`finalize()`, deprecato dalla versione 9 e sostituito con metodi più flessibili) che viene invocato nel momento in cui un oggetto viene deallocato dal GarbageCollector

Esercizio 1

- Creare una classe Java per descrivere un *rettangolo*, che ha una *larghezza* e un'*altezza* (per semplicità, valori interi) e del quale si vuole poter calcolare *perimetro* e *area*.

- Istanziare due rettangoli **`r1`** ed **`r2`** che hanno, rispettivamente, le seguenti dimensioni:
  - **`r1: larghezza: 20 e altezza: 50`**
  - **`r2: larghezza: 80 e altezza: 10`**

Esempio rettangolo (in C)

```
class rettangolo{
private: int l, h;

public:
rettangolo(int len, int he){l=len;h=he;};

int perimetro() {return (2*(l+h));}

int area() {return (l*h);}
};
```

Esempio rettangolo (in Java)

```
public class Rettangolo {
        private int l;
        private int h;

        public Rettangolo(int l, int h) {
                this.h = h;
                this.l = l;
        }

        int getPerimetro(){
                return (this.l+this.h)*2;
                }

        int getArea(){
                return (this.l*this.h);
                }
}
```

Tornando sui tipi: wrapper e automatic (un)boxing

- Ci sono diverse strutture dati ed algoritmi in Java che lavorano su oggetti e non su tipi primitivi
- Java definisce una classe *wrapper* per ogni tipo primitivo
- Fornisce inoltre meccanismi automatici di conversioni tra i due

Base Type	*Class Name*	*Creation Example*	*Access Example*
boolean	Boolean	obj = new Boolean(true);	obj.booleanValue()
char	Character	obj = new Character('Z');	obj.charValue()
byte	Byte	obj = new Byte((byte) 34);	obj.byteValue()
short	Short	obj = new Short((short) 100);	obj.shortValue()
int	Integer	obj = new Integer(1045);	obj.intValue()
long	Long	obj = new Long(10849L);	obj.longValue()
float	Float	obj = new Float(3.934F);	obj.floatValue()
double	Double	obj = new Double(3.934);	obj.doubleValue()

Altri tipi comuni: String ed Enumeration

- Java ha una classe built-in per lavorare sulle stringhe: `String`
- `String` è usato quindi per memorizzare sequenze di caratteri, eventualmente vuote
- Sulle istanze di `String` è possibile fare operazioni comuni:
  - concatenazione: operatore `+`
  - calcolo della lunghezza: `.length()`
  - estrazione di sottostringhe: `substring(…)`
  - etc.

- Le enumerazioni sono usate per rappresentare scelte tra un set finito di valori

```
public enum Day { MON, TUE, WED, THU, FRI, SAT, SUN };
```

Esercizio 1b

- Implementare in Java una classe Persona, caratterizzata da: nome, cognome e nazionalità, dove nazionalità può essere scelta tra ‘Italiana’, ‘EU’, ‘Extra EU’.

Packages

- Java usa un approccio semplice ma funzionale per organizzare i file e le classi di un programma: ogni classe è memorizzata in un file separato, con lo stesso nome ed estensione `.java`
- Classi e definizioni (es. enumerazioni) diverse possono essere raggruppate in **packages**
- I nomi dei package sono solitamente in minuscolo e tutti i file appartenenti allo stesso package sono nella stessa directory e contengono l'istruzione

```
package nomePackage;
```

- I package sono organizzati in subpackage con una struttura gerarchica

Import e accesso ai package

- Per fare riferimento ad un nome in un package si usa un nome qualificato tramite l'operatore .

```
java.util.Scanner(…)
```

- Per includere classi da un altro package si usa la keyword import

```
import java.util.Scanner;
 ...
 Scanner input = new Scanner(System.in);
```

- Si può importante un intero package se ci si aspetta di usare molte classi (più dispendioso)

```
import java.util.*;
```

Esercizio 2

- Scrivere una classe Java per gestire un conto corrente, su cui sono possibili le seguenti operazioni:
  - Apri conto corrente vuoto
  - Apri conto corrente con X euro
  - Versa X euro nel conto
  - Preleva X euro dal conto
  - Stampa un messaggio con il salto attuale

Ereditarietà

- Java (in generale OOP) permette di a partire da un'altra, dalla quale si **ereditano** campi e metodi (protected e public)
  - La classe che eredita si chiama classe *derivata*, o *sottoclasse*, o classe *figlio*
  - La classe di partenza è la classe *base*, o classe *padre*, o *superclasse*
- La modifica di un metodo della classe padre si chiama **overriding**
- Java usa la keyword `extends` per indicare ereditarietà
- La keyword `super` permette di accedere alla classe padre
- Java non supporta ereditarietà multipla

Esercizio 3

- Estendere la classe precedente per poter gestire conti con valute diverse
- Nuove proprietà e metodi:
  - `valuta`: campo privato che può assumere valori “euro”, “dollar”, “pound”
  - Costruttore `ContoCorrenteConValuta(Integer s, Valuta v)`: prende in input il saldo iniziale la valuta
  - `mostraSaldo()`: override del metodo precedente per stampare il saldo in una data valuta (es. “Nel c/c hai 100 euro”)

Ereditarietà e visibilità

E’ corretto sovrascrivere `mostraSaldo()` in questo modo?

```
public void mostraSaldo(){
        System.out.print("\nC/C con " + this.saldo +
        this.valuta);
        }
```

Si può fare in altro modo?

Interfacce

- Il paradigma OOP permette di specificare chiaramente le interfacce o **API** (**Application Programming Interface**) di una classe
- Tramite **Strong Typing** si verifica in fase di compilazione che le interfacce e i vincoli sui tipi siano rispettati
- In Java la keyword `implements` permette di imporre l'implementazione di una o più interfacce da parte di una classe
- Un'interfaccia è un insieme di dichiarazioni di metodi senza dati e senza implementazione
- Una classe può implementare più interfacce

Esercizio 4

- Implementare le classi Java necessarie a modellare questa situazione:

- *Ogni animale fa un verso e ha un certo numero di zampe*
- *Il gatto ha 4 zampe e miagola*
- *Il cane ha 4 zampe e abbaia*
- *Il tacchino ha 2 zampe e goglotta*

- Usare un'interfaccia che descrive le due operazioni:
  - fai un verso
  - dimmi quante zampe hai

Esercizio 5

- Definire in Java un'interfaccia *Poligono* che ha i metodi:
  - `getArea()`
  - `getPerimetro()`
- Per semplicità tutte le dimensioni sono espresse con numeri interi

- Implementare le seguenti classi, ognuna con il proprio costruttore (che prende in input le dimensioni dei lati) e i due metodi precedenti:
  - `Quadrato(lato)`
  - `Rettangolo(larghezza, altezza)`