Rassegna API - 1

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/mizzaro/
mizzaro@dimi.uniud.it

Programmazione, lezione 20
25 gennaio 2007

Riassunto

- Programmazione strutturata
- TDA
  - Classi, incapsulamento, …
- OO
  - Scambio messaggi, eredità, polimorfismo
- OO in Java
  - Classi astratte, interfacce, classi interne
  - Upcast, downcast

Scaletta

- Fine OO in Java: Package
- Inizio rassegna API
  - `Math`
  - `Object`
  - `System`
  - `String`

Package

- Insieme di classi e interfacce
- "Namespace"
- Struttura gerarchica
- Basata sul filesystem
- Nomi univoci grazie al dominio internet
- Due istruzioni (parole riservate):
  `package` e `import`

`package`

- In quale package mettere le classi (e le interfacce) del file
  `package <nomepackage>;`
- Prima istruzione di un file
- Se l'istruzione non c'è (come finora) ⇒ package "di default"

`import`

- Dice in quali package/classe cercare i nomi usati nelle classi di questo file
  `import <nomepackage>.<nomeclasse>;`
  `import <nomepackage>.*;`
- Esempi
  - `import java.util.Calendar;`
  - `import java.util.GregorianCalendar;`
  - `import java.util.*;`
- N.B.: si importa (tutte le classi di) un package o specifiche classi, non i "sottopackage":
  `import java.*;` è sbagliato

Full qualified name

- ***<nomepackage>.<nomeclasse>***
- **`java.util.Calendar`**
- **`java.lang.String`**
- **`javax.swing.JApplet`**
- Quindi: l'**`import`** serve per evitare di scrivere i full qualified name ogni volta
  - In realtà non si importa nulla: si evita di scrivere

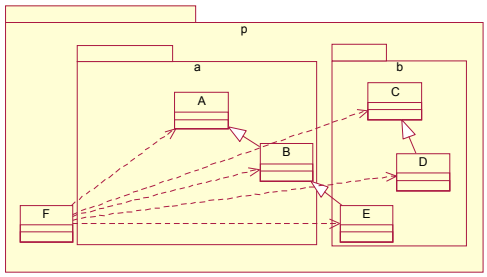
Package e filesystem

- La struttura gerarchica dei package corrisponde a quella del filesystem
  - Le classi del package **`a`** vanno nella directory **`a`**
  - Le classi del package **`a.b`** vanno nella directory **`a/b`**
  - …
  - I file del package **`xxx.yyy.zzz`** vanno nella directory **`xxx/yyy/zzz`**
  - E la radice deve essere in **`CLASSPATH`**

Esempio

- Nel package **`p.a`** (directory **`./p/a`**) ci sono 2 classi **`A`** e **`B`** (sottoclasse di **`A`**)
- Nel package **`p.b`** (directory **`./p/b`**) ci sono 3 classi **`C`**, **`D`** (sottoclasse di **`C`**) ed **`E`** (sottoclasse di **`B`**)
- Nel package **`p`** (directory **`./p`**) c'è la classe **`F`** (che usa tutte le altre classi)

Diagramma dei package



Package, compilazione, esecuzione, javadoc

- Bisogna risalire alla radice
- Per compilare (usare nomi di file)
  - **`>javac -d bin p/F.java`**
  - **`>cd p; javac F.java`** (non trova i package)
- Per eseguire (usare full qualified names)
  - **`>java p.F`**
  - **`>cd p; java F`** (non trova la classe **`F`**, vuole **`p.F`**)
  - **`>cd p; java p.F`** (non va, cerca il package/dir **`p`**)
- Javadoc (creazione documentazione):
  - **`>javadoc -verbose -d doc p p.a p.b`**
  - (Sempre risalire alla radice)

Nomi univoci di package

- Nome di package: **`xxx.yyy.zzz`** (minuscole!)
- Nomi che iniziano con **`java.`** sono riservati (**`java.lang`**, **`java.io`**, **`java.util`**, …)
- Dal nome del dominio internet: il nome di un package sviluppato in **`uniud.it`** inizia con **`it.uniud`**
- (e prosegue con un nome esplicativo deciso dal programmatore: **`it.uniud.sindy`**, **`it.uniud.twm`**, …)

Package e visibilità

- Attributi e metodi possono avere visibilità:
  - **`public`**: ovunque, anche in altri package
  - **`protected`**: solo nelle sottoclassi e nel package
  - (niente) "package": solo nel package
  - **`private`**: solo nella classe
- Classi e interfacce possono avere visibilità:
  - **`public`**: anche da altri package
  - (niente) "package": solo nel package
- Regola: un'unica classe **`public`** in un ".java", e nome classe = nome file

Scaletta

- Package
- Inizio rassegna API
  - **`Math`**
  - **`Object`**
  - **`System`**
  - **`String`**

Importanza rassegna API

- Rassegna API?
  - Java 1.4.2: 2991 classi in 135 package! Gulp!!
  - Java 5 (1.5): di più!!!
- Non riusciamo a vedere tutte le API in qs. corso ma
  - Da qualche parte bisogna pur cominciare
  - È importante capire il modo "giusto" di lavorare di un programmatore moderno (che cerca pezzi di codice e li "incolla")
- È importante evitare di ri-inventare la ruota
- Molto di quello che deve fare un programmatore è già fatto,
  - Si tratta "solo" di trovarlo…
  - È anche importante capire come e perché è fatto…

Fonti utili

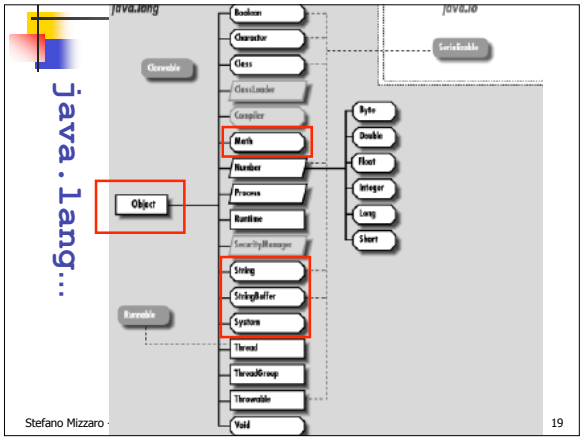
- Documentazione in linea delle API
  - Creata con javadoc
  - In formato HTML
  - Navigabile con un browser
  - (…java/docs/)
- Sorgenti (.java) delle API!
  - File src.zip nell'SDK, da scompattare
  - (…java/src/…)

Un po' di storia…

Anno	Vers.	# classi	# package	Altro
1991	Oak			Elettrodomestici, …
1995	α			HotJava, applets
1996	1.0	212	8	Web browser
1997	1.1	504	23	
1998	1.2	1520	59	Java 2: Swing, collections
2000	1.3	1842	76	Bug fix, HotSpot
2003	1.4	2991	135	NIO, pattern match & r.e., XML (DOM/SAX), assert, …
2005	1.5 (5.0)	~3279	~166	Generics, autoboxing, for, #args variabile, …

Un po' di package usati spesso

- **`java.lang`**: (importato implicitamente)
- **`java.util`**: (date, contenitori di oggetti, …)
- **`java.io`**: input/output, flussi, file, …
- **`java.awt`**, **`java.awt.event`**, **`javax.swing`** (GUI)
- **`java.applet`** (applet in pagine Web)
- **`java.sql`** (JDBC x accesso DB)
- **`java.rmi`** (Remote Method Invocation)
- …
- Un po' di **`java.lang`**: **`Math`**, **`Object`**, **`System`**, **`String`**



java.lang.Math

```
public static double sin(double)
public static double cos(double)
public static double sqrt(double)
public static double log(double)
public static double abs(double)
public static double exp(double, double)
public static double floor(double)
public static double ceil(double)
public static double min(double, double)
public static double max(double, double)
...
```

- Se vi serve una funzione matematica → documentazione API…

java.lang.Object

- Radice della gerarchia, superclasse di tutte le classi
  - Se non c'è **`extends`**, è come se ci fosse **`extends Object`**
- I suoi metodi sono ereditati da tutte le classi
  - **`public String toString()`**
  - **`public boolean equals(Object o)`**
  - (**`protected Object clone()`**)
  - …

toString()

- **`public String toString()`**
- Restituisce una rappresentazione testuale, visualizzabile dell'oggetto
- Chiamato implicitamente quando serve
- Da sovrascrivere in ogni classe che definiamo
- (vediamo com'è fatto…)

Esempio

```
class Punto {
  // ...
  public String toString () {
    return "Punto: (" + getX() + ", " +
           getY() + ")";
  }
}
```

```
class ProvaPunto {
  // ...
  public static void main(String[] args) {
    Punto p = new Punto(2.3, 5.67);
    System.out.print(p);
    System.out.println(p);
  }
}
```

```
>java ProvaPunto
Punto: (2.3, 5.67)…
```


equals

- **`public boolean equals(Object o)`**
- 2 tipi di uguaglianza fra **`x`** e **`y`**:
  - Essere lo stesso oggetto (**`x == y`**)
  - Essere 2 oggetti uguali (**`x.equals(y)`**)
- **`==`** confronta i riferimenti
- In realtà, l'**`equals`** di **`Object`** è come l'**`==`**…
- …ma **`equals()`** può essere sovrascritto
- Da sovrascrivere in ogni classe che definiamo

Esempio (1/3)

```
class Punto {
  // ...
  public boolean equals (Object o) {
    return ((Punto)o.x == this.x &&
            (Punto)o.y == this.y);
  }
}
```

- Notate il downcast: devo sovrascrivere, e il parametro di **`equals`** è **`Object`**…
- Si può fare di meglio…

Esempio (2/3)

```
class Punto {
  // ...
  public boolean equals (Object o) {
    if(o instanceof Punto)
      return ((Punto)o.getX() == this.getX()
                &&
              (Punto)o.getY() == this.getY());
    else
      return false;
  }
}
```

- Ancora meglio…

Esempio (3/3)

```
class Punto {
  // ...
  public boolean equals (Object o) {
    return (o instanceof Punto &&
            ((Punto)o).getX() == this.getX()
            &&
            ((Punto)o).getY() == this.getY()
           );
  }
}
```


Scaletta

- Package
- Inizio rassegna API
  - **`Math`**
  - **`Object`**
  - **`System`**
  - **`String`**

`java.lang.System`

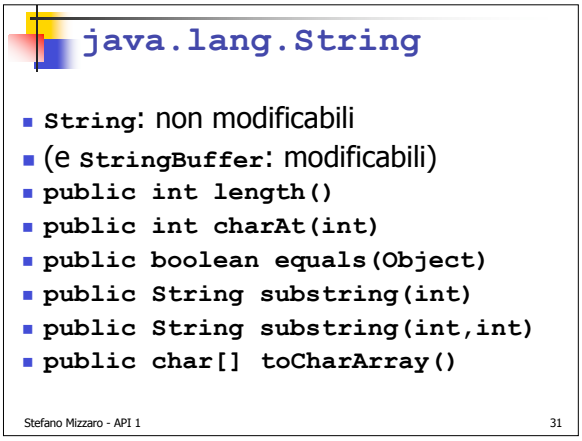
- Sveliamo un mistero: perché
  - **`System.out.println`** (**`print`**)
  - **`System.in.read`**
- Classe **`System`**
- Variabile di classe **`out`** (**`in`**)

```
public final static PrintStream out;
```

- Metodi d'istanza **`print, println, read`**

`java.lang.String`

- Sequenza di caratteri
- **`String`**: sottoclasse **`final`** di **`Object`**
- "Zucchero sintattico":
  - Letterali stringa: **`""`** (chiamata implicita del costruttore)
  - **`+`**: concatena stringhe


java.lang.String
String: non modificabili
(e StringBuffer: modificabili)
public int length()
public int charAt(int)
public boolean equals(Object)
public String substring(int)
public String substring(int,int)
public char[] toCharArray()
Stefano Mizzaro - API 1
31


Riassunto
Package
Inizio rassegna API
Math
Object
System
String
Stefano Mizzaro - API 1
32