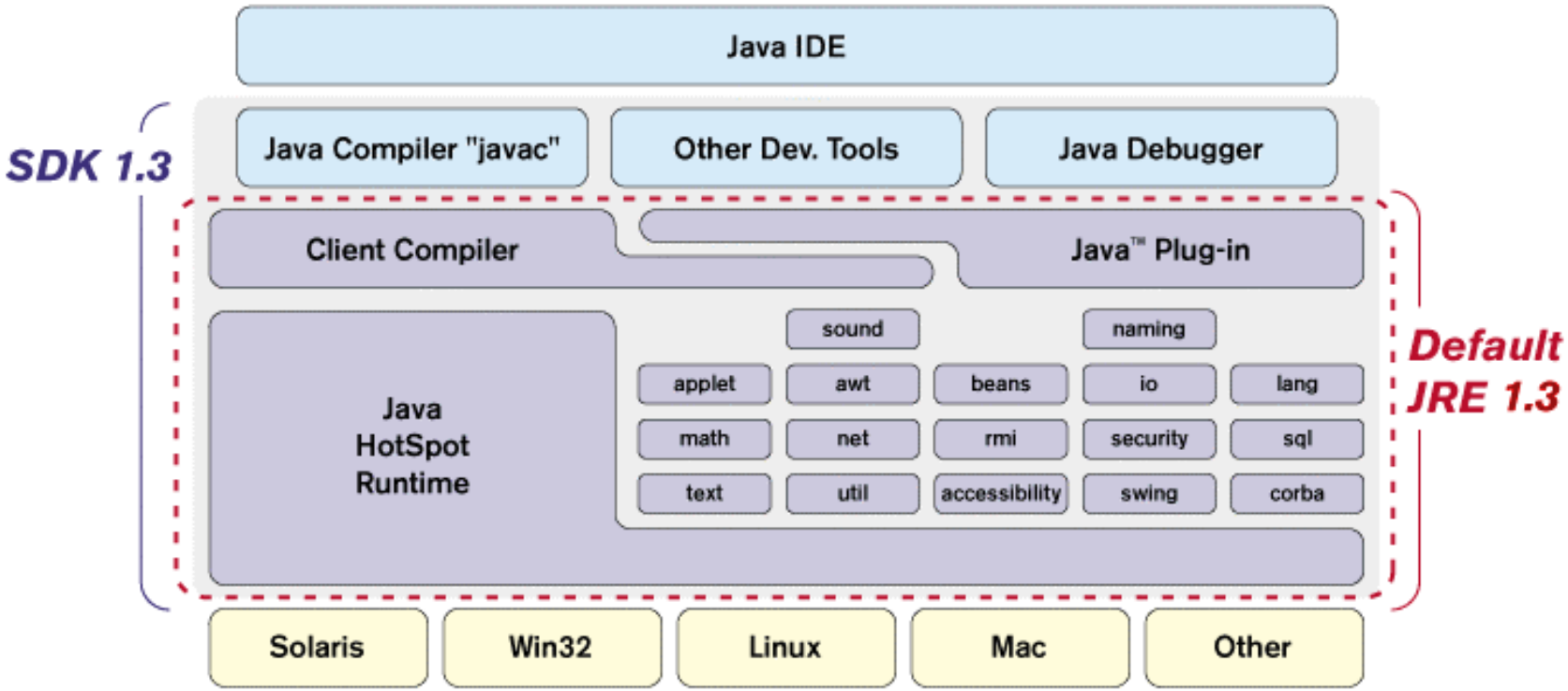
DAY 1.1

Ambiente Java



Compilatori

- Sorgente -> bytecode
  - portabile
  - completamente decompilabile
  - interpretato
- Sorgente -> eseguibile nativo
  - non portabile
  - non facilmente decompilabile
  - piu’ veloce di C e C++

Bytecode e Interprete

- Intestazione fornisce versione
- Istruzioni della macchina virtuale
- Lunghezza constante
- Stack machine tipata

JIT compiler

- JIT == Just In Time
- Compila il byte code al tempo dell'esecuzione
- produce codice macchina nativo (non portabile)
- migliora il tempo di esecuzione (tranne che la prima volta)
- Non effettua ottimizzazioni globali

Direttive di programma

- una o piu’ linee
- un token non puo’ stare su piu’ di una linea
- terminate da ;
- System.out.println(“HELLO”);
  - equivale a
- System
- .out
- .println
- (“HELLO”
-);

Direttive di programma (Cont.)

- Ma non
- Sys
- tem.out.pr
- intln(“HELLO”);

- Comunque attenzione alla leggibilita’

Concetti base di programmazione a oggetti

- Costruzione dei tipi
- Oggetti e istanze
- Incapsulazione delle caratteristiche
- Derivazione e specializzazione
- Manipolabilita’

Classi

- Definisce un oggetto elencandone
  - le caratteristiche
  - le possibili manipolazioni
  - le relazioni con altre classi (Classi derivate)

Classi derivate

- Java permette di definire classi derivate da altre classi
- Una classe derivata specifica ulteriormente la classe da cui deriva
- Una classe puo’ derivare da una sola altra classe

Tipi di dati

- Oggetti
  - definiti da classi
  - passati sempre per riferimento
- tipi primitivi
  - passati sempre per valore

La libreria di classi di SDK 1.2

- Stesso principio di C e C++
- Non keyword come in pascal
- librerie standard
  - I/O
  - Networking
  - Utilities
  - Reflections
  - SQL (JDBC)
  - ...

Tipi primitivi e classi relative

- boolean <-> Boolean
- byte <-> Byte
- short <-> Short
- int <-> Integer
- long <-> Long
- float <-> Float
- double <-> Double

Aritmetica e Regole di conversione

- Aritmetica intera sempre almeno a 32 bit
- Upcast automatico
  - byte->short->int->long->float->double
- Downcast esplicito
  - short a,b,c;
  - a = b + c; // non compila
  - a = (short)(b + c) // ok
- ECCEZIONE operatori op=

Operatori e loro precedenza

- Le stesse del C e C++

(), [], ., X++, X--

+ unario, = unario, ++X, --X, ~, !

Cast esplicito, new

*, /, %

+, -

<<, >>, >>>

<, <=, >, >=, instanceof

==, !=

&

^

|

&&

||

?:

=, op=

- CONSIGLIO: se avete dubbi usate le parentesi

Commenti e documentazione

- come c++
  - /* …… */
  - // sino a fine linea
- Commenti per documentazione automatica
- /**
- * Questo commento viene usato per creare
- * automaticamente la documentazione della
- * classe.
- */

Confronti

- Confronti come nel C e C++
- >
- >=
- ==
- !=
- <=
- <

Costrutti di controllo

if(<boolean value>)

{

}

else

{

}

- Indentazione non codificata, dipende dallo stile personale

Costrutti di controllo (Cont.)

```
if(<boolean value>)
{
    if(<boolean value>)
    {
    }
    else
    {
    }
}
else if(<boolean value>)
{
}
else if(<boolean value>) …. Ad libitum
```

Costrutti di controllo (Cont.)

- Esiste l’operatore condizionale
- a > b ? C : D;
- Se volete partecipare al campionato “Offuscated Java code” ….

Costrutti di controllo (Cont.)

```
switch(<expression>)
{
    case <value>:
        break;
    case <value>:
        break;
    case <value>:
        // fallthrough!!
    case <value>:
        break;
    default:
        break;
}
```

Costrutti di controllo (Cont.)

for(<inizializzazione>;<condizione di uscita>;<azione iterativa>)
{
    <corpo>
}

- **equivale**

<inizializzazione>
while(!<condizione di uscita>)
{
    <corpo>
    <azione iterativa>
}

Costrutti di controllo (Cont.)

```
do
{
    <corpo eseguito almeno una volta>
}
while(<condizione di continuazione)
```

Controllo dei loop

continue

- passa direttamente all'iterazione successiva senza eseguire il resto del corpo
- la condizione iterativa di un loop for viene eseguita
- la condizione iterativa del loop while equivalente NON viene eseguita

break

- interrompe l'esecuzione del loop e passa alla prima istruzione fuori dal loop

Controllo dei loop (Cont)

- continue e break possono usare etichette

```
Uscita:
for(i=0;i<1000;i++)
    for(j=0;j<1000;j++)
    {
        <corpo 1>
        if(condizione eccezionale)
            continue Uscita;
        <corpo 2>
        if(condizione di errore)
            break Uscita
    }
```

Array

- Array sono veri oggetti, non solo un modo conveniente di indicizzare una zona di memoria come in C o C++
- Array in Java hanno una lunghezza definita
  - non si puo' sforare e massacrare i dati senza saperlo
  - overhead dovuto al controllo degli indici
- Dichiarazione != definizione
- Definizione del nome
- dichiarazione alloca la memoria

Array (Cont.)

- int pippo[][];
- pippo = new int[10][];
- pippo[0] = new int[0];
- pippo[1] = new int[1];
- ...
- pippo[9] = new int[9];

- long pluto[][][];
- etc.

Stringhe

- Le stringhe sono oggetti immutabili come le classi corrispondenti ai tipi primitivi
  - non ci sono problemi di sforamento della memoria allocata
  - overhead di gestione degli oggetti
- Si opera su stringhe come oggetti tramite metodi specifici

Stringhe (Cont)

char charAt(int index)

Returns the character at the specified index.

int compareTo(Object o)

Compares this String to another Object.

int compareTo(String anotherString)

Compares two strings lexicographically.

int compareToIgnoreCase(String str)

Compares two strings lexicographically, ignoring case considerations.

Stringhe (Cont)

String concat(String str)

Concatenates the specified string to the end of this string.

static String copyValueOf(char[] data)

Returns a String that is equivalent to the specified character array.

static StringcopyValueOf(char[] data, int offset, int count)

Returns a String that is equivalent to the specified character array.

Stringhe (Cont)

boolean endsWith(String suffix)

Tests if this string ends with the specified suffix.

boolean equals(Object anObject)

Compares this string to the specified object.

boolean equalsIgnoreCase(String anotherString)

Compares this String to another String, ignoring case considerations.

Stringhe (Cont)

byte[] getBytes()

Convert this String into bytes according to the platform's default character encoding, storing the result into a new byte array.

byte[] getBytes(String enc)

Convert this String into bytes according to the specified character encoding, storing the result into a new byte array.

Stringhe (Cont)

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

Copies characters from this string into the destination character array.

int hashCode()

Returns a hashcode for this string.

int indexOf(int ch)

Returns the index within this string of the first occurrence of the specified character.

Stringhe (Cont)

int indexOf(int ch, int fromIndex)

Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

int indexOf(String str)

Returns the index within this string of the first occurrence of the specified substring.

int indexOf(String str, int fromIndex)

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Stringhe (Cont)

String intern()

Returns a canonical representation for the string object.

int lastIndexOf(int ch)

Returns the index within this string of the last occurrence of the specified character.

int lastIndexOf(int ch, int fromIndex)

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

Stringhe (Cont)

int lastIndexOf(String str)

Returns the index within this string of the rightmost occurrence of the specified substring.

int lastIndexOf(String str, int fromIndex)

Returns the index within this string of the last occurrence of the specified substring.

int length()

Returns the length of this string.

Stringhe (Cont)

boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)

Tests if two string regions are equal.

boolean regionMatches(int toffset, String other, int ooffset, int len)

Tests if two string regions are equal.

String replace(char oldChar, char newChar)

Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

Stringhe (Cont)

boolean startsWith(String prefix)

Tests if this string starts with the specified prefix.

boolean startsWith(String prefix, int toffset)

Tests if this string starts with the specified prefix beginning a specified index.

String substring(int beginIndex)

Returns a new string that is a substring of this string.

String substring(int beginIndex, int endIndex)

Returns a new string that is a substring of this string.

Stringhe (Cont)

char[] toCharArray()

Converts this string to a new character array.

String toLowerCase()

Converts all of the characters in this String to lower case using the rules of the default locale, which is returned by Locale.getDefault.

String toLowerCase(Locale locale)

Converts all of the characters in this String to lower case using the rules of the given Locale.

Stringhe (Cont)

String toString()

This object (which is already a string!) is itself returned.

String toUpperCase()

Converts all of the characters in this String to upper case using the rules of the default locale, which is returned by Locale.getDefault.

String toUpperCase(Locale locale)

Converts all of the characters in this String to upper case using the rules of the given locale.

Stringhe (Cont)

String trim()

Removes white space from both ends of this string.

static String valueOf(boolean/char/char[]/double/float/int/long/byte/Object b)

Returns the string representation of the boolean/char/char[]/double/float/int/long/byte/Object argument.

Stringhe (Cont)

static String valueOf(char[] data, int offset, int count)

Returns the string representation of a specific subarray of the char array argument.

StringBuffer

- Per migliorare l’efficienza si puo’ usare il tipo StringBuffer
- oggetti che possono essere cambiati
  - risparmia overhead di creazione di nuovi oggetti
- Capacita’ uguale alla lunghezza della string contenuta + 16 caratteri Unicode
- La capacita’ puo’ essere cambiata a runtime

StringBuffer (Cont)

- La lunghezza puo’ essere cambiata a runtime troncando la stringa
- Si possono cambiare i singoli caratteri come per una stringa in C o C++

```
StringBuffer sb = new
   StringBuffer(“cane”);
sb.setCharAt(0, ‘p’);
```

DAY 1.2

Definizione di classi

- Definire un tipo di oggetto e la semantica che gli si vuole dare tramite
  - campi
    - come i campi di una struttura C
  - metodi
    - non esiste parallelo diretto in C
  - blocchi di inizializzazione
    - eseguiti al caricamento, prima di generare qualunque istanza della classe, una sola volta
- I nomi delle classi cominciano con la maiuscola

Classe DisneyCharacter

```
package it.unige.dist.laser.esempio;
public class DisneyCharacter
{
        int yob = -1;
        String name;
        protected DisneyCharacter(String theName, int theYob)
        {
              name = theName;
              yob = theYob;
        }
        public String isFunny()
        {
                return “tutti i personaggi Disney sono divertenti”;
        }
        public String toString()
        {
              return new String(name + “ “ + String.valueOf(yob));
        }
        static
        {
                System.out.println(“Hurra!”);
        }
}
```

Campi e tipi di campi

- campo di istanza
  - uno per ogni istanza della classe
  - tipo di default
- campo di classe
  - uno comune a tutte le istanze della classe
  - richiede la keyword “static”

Metodi e tipi di metodi

- metodo di istanza
  - uno per ogni istanza della classe
  - tipo di default
- metodo di classe
  - uno comune a tutte le istanze della classe
  - richiede la keyword “static”

Passaggio parametri

- parametri formali e parametri attuali
- tipi primitivi per valore
- Oggetti per indirizzo
  - si possono definire “final” e non possono essere cambiati (controllo a tempo di compilazione)

Accesso a campi e metodi

- Un campo o un metodo di istanza richiede di avere una istanza attiva della classe per potervi fare riferimento

  String s = new String(“pippo”);

  s.length(); // metodo di istanza

- Un campo o metodo di classe si riferisce al nome della classe stessa

  int i = 10;

  String.valueOf(i);

Regole di visibilita’

- Private
- Protected
- “Default”
- Public

Costruttori

- I costruttori hanno il nome della classe
- I costruttori non definiscono il tipo di ritorno
- Il costruttore di default (senza parametri) se non definito esplicitamente viene generato automaticamente dal compilatore
- Un construttore di default generato dal compilatore non fa nulla (a parte invocare il costruttore di default della classe madre se esiste)

garbage collection

- Java non ha deallocazione esplicita
- un oggetto rimane in memoria almeno sino a che esiste un riferimento ad esso
  - in pratica una volta perso l'ultimo riferimento puo' passare del tempo
- non ci possono essere memory leak causati da programmi utente
- la garbage collection introduce due overhead
  - contare i riferimenti
  - effettuare la garbage collection vera e propria

Overloading dei metodi

- Si possono avere piu' metodi con lo stesso nome purche' la signature (numero e/o tipo dei parametri formali) sia diversa
- Il valore di ritorno non e' sufficiente a dstinguere

```
public class Esempio
{
    int value(int a) {};
    int value(long a) {};
    long value(int a, long b) {};
    long value(long a) {}; //errore al tempo di compilazione!
}
```

Ereditarieta’

- Una classe puo’ estendere un’altra classe
- Eredita cio’ che era gia’ definito nella classe madre
  - campi e metodi
- il processo e’ addittivo
  - a estende b estende c
  - implica che a contiene tutto cio’ che e’ in b e in a
- un tipo derivato e’ sempre anche di ogni tipo piu’ astratto, non e’ vero il contrario
  - a e’ di tipo a, di tipo b e di tipo c
  - b e’ di tipo b e di tipo c
  - c e’ solo di tipo c

Esempio classe derivata

```
package it.unige.dist.laser.esempio;
public class Pippo extends DisneyCharacter
{
	public pippo()
	{
		super(“Pippo”, 1932) // 25 maggio, per essere precisi
	}
	public String isFunny()
	{
	    return “tra i personaggi Disney Pippo e’ uno dei piu’ divertenti”;
	}
	static
	{
		System.out.println(“Yuk yuk!”);
	}
}
```

Classi astratte

- Una classe puo' essere "astratta", cioe' definire la necessita' di avere un tipo di comportamento al fine di appartenere ad una categoria logica

```
public abstract class Astratta
{
    public abstract int quanti();
}
```

  - non puo' essere instanziata direttamente
  - deve essere estesa e la classe derivata deve implementare tutti i metodi astratti della classe base

Polimorfismo

- La capacita’ identificare solo a runtime l’effettivo comportamento del programma

```
Public static void main(String[] argv)
{
    Astratta a = <leggi da canale di input>;
    Sstem.out.println(String.valueOf(a.quanti()));
}
```

Overriding dei metodi

- Si usa per implementare il polimorfismo
- I metodi in Java sono virtuali per default
- il metodo invocato sara' sempre quello della classe piu' derivata possibile purche'
  - il metodo invocato deve essere presente anche nella classe base
  - la signature del metodo deve essere la stessa in classe base e classe derivata INCLUSO IL TIPO RITORNATO
- non e' possibile restringere l'accesso a un metodo derivato

Modificatore final

- serve per prevenire la modificazione della semantica di una classe
- i metodi di Java sono “virtuali”
- se io derivo una classe viene eseguito il codice della classe derivata
- posso alterare la semantica di un oggetto ridefinendone i metodi in una classe derivata
- se la classe e’ definita “final” questo e’ impossibile
- e’ possibile anche definire final singoli metodi o campi

Interfacce

- Guerra di religione Distributed OOP:
- Implementation Inheritance vs. Delegation and Containement
- Derivazione da classi base vs. Definizione di Interfacce e loro implementazione

Uso delle Interfacce

contratto tra utilizzatore e fornitore di servizi

permette di nascondere completamente l'implementazione e cristallizzare la modalita' di interazione

metodologia usata pesantemente in quasi tutti gli schemi di distributed OOP (e.g. CORBA, RMI e DCOM)

ATTENZIONE: contratto sintattico, non si dice nulla in proposito alla semantica!!!

Definire un’Interfaccia

```
public interface Button
{
    public String getName();
    public void press();
    public boolean isPressed();
}
```

Implementare un'Interfaccia

```
public class MyButton implements Button
{
        String name = "Innominato";
        boolean state = false;
        public MyButton(String theName)
        {
                name = theName;
        }

        public String getName()
        {
                return new String("Il mio bottone si chiama " + name);
        }
        public void press()
        {
                state ^= true;
        }
        public boolean isPressed()
        {
                return state;
        }
}
```

Concetto e uso dei Package

- package == insieme di classi legate da un comune attributo
  - implementano un servizio
  - sono logicamente simili e.g. tutte le classi che implementano un formato di immagine
  - ...
  - stesso programmatore?
- sono l’unita’ di visibilita’ di default

Concetto e uso dei Package (Cont)

- I nomi dei package cominciano con la minuscola
- I nomi dei package dovrebbero seguire la convenzione contraria a quella degli URL

package it.unige.dist.laser.pippo;

package com.sun.pluto;

import java.util.*;