Variabili e Oggetti

Lo spazio di memoria di Java

Le API di Java

Usare e costruire oggetti

- Gli oggetti sono entità di un programma che si possono manipolare invocando i metodi
- `System.out` oggetto della classe `PrintStream`
- `println(String s)` metodo della classe `PrintStream`
- `System.out.println("Hello, World!");` invocazione del metodo `println` sull'oggetto `System.out` passando come parametro l'oggetto `String "Hello, World!"`

Interfaccia Pubblica di una classe

- Specifica cosa si può fare con gli oggetti che si creano da essa
- Contiene tutti i campi e i metodi public definiti all'interno della classe
- Rappresenta "il contratto" della classe: cosa si impegna a fare
- È la candidata naturale a "documentazione della classe"
- Le classi standard fornite con la SDK hanno la documentazione sull'interfaccia pubblica

Application Program Interface

- A.P.I. E' la documentazione riguardande le interfacce pubbliche di tutte le classi (e altro) fornite con la SDK
- Online sul sito della sun (link alle API delle varie distribuzioni):

  http://java.sun.com/reference/api/index.html
- Scaricabile anche come .zip per averlo in locale su

  http://java.sun.com/docs/index.html

Esempio: la classe String

- L'interfaccia pubblica della classe String specifica l'esistenza di un metodo di nome length

```
int    length()
```

Returns the length of this string

- Restituisce un numero intero (`int`) che rappresenta la lunghezza
- `System.out.println("Hello, World!".length());`
- Stampa 13, il numero di caratteri nella stringa

Esercizio

- Trovare la documentazione della classe `PrintStream`
- Trovare la documentazione della classe `String`
- Cercare di interpretare, per quanto possibile, tutte le sezioni della documentazione
- Estendere `Hello.java` usando qualche metodo nuovo sia della classe `String` che della classe `PrintStream`

Creazione di Oggetti

- `System.out` è già creato automaticamente dalla JVM
- `"Hello, World!"` come oggetto della classe `String` viene creato intrinsecamente visto che è inserito nel testo del programma
- Ma il modo principale per creare oggetti è quello di usare l'operatore `new`

```
new Rectangle(5,10,20,30)
```

- Crea un oggetto della classe Rectangle

L'oggetto creato

- Rectangle è una classe inclusa nella SDK. Nelle API:

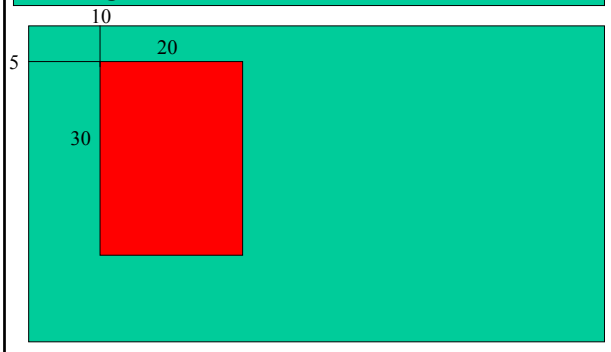
`Rectangle(int x, int y, int width, int height)`

Constructs a new Rectangle whose top-left corner is specified as (x, y) and whose width and height are specified by the arguments of the same name.

Costruzione

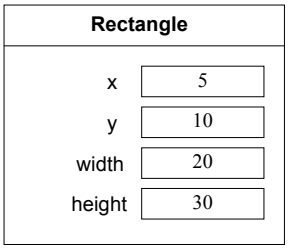
- Il processo di creazione di un nuovo oggetto
- Dopo la parola chiave new va inserito specificato il **costruttore** per l'oggetto
- I costruttori sono metodi speciali che
  - Hanno come nome lo stesso nome della classe (in questo caso `Rectangle`)
  - Hanno diversi parametri di costruzione
- Nel nostro caso abbiamo usato il costruttore, indicato nelle API, che prende 4 interi

Rectangle: il concetto



Rectangle: l'oggetto

Heap



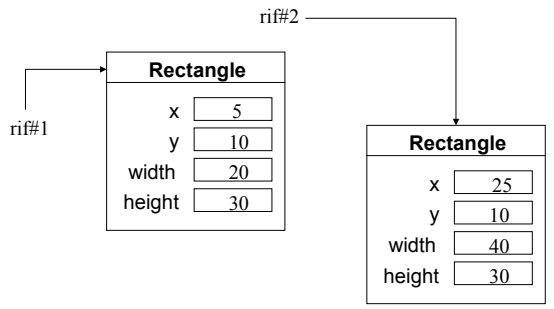
Lo spazio di memoria del linguaggio Java

- La memoria è idealmente divisa in due sezioni:
  1. Lo heap (mucchio) che contiene gli oggetti
  2. Lo stack di attivazioni che contiene le variabili del programma

Lo Heap

- È uno spazio di memoria gestito automaticamente e completamente dalla JVM
- Contiene gli oggetti creati dal programma
- Ogni oggetto residente ha un indirizzo, o riferimento, che lo identifica univocamente e lo rende raggiungibile dal programma
- Gli oggetti non più riferiti da nessuna variabile vengono "cancellati": lo spazio che occupano viene dichiarato libero ed è riutilizzabile

Lo Heap



Lo stack di attivazioni

- Le variabili dichiarate nel programma sono conservate in una zona apposita di memoria
- Questa zona è strutturata, a differenza dello heap
- La struttura è una pila (stack) di "attivazioni"
- Un'attivazione consiste di un particolare contesto di esecuzione
- Il metodo main è l'attivazione principale e si trova sempre in fondo alla pila

Lo stack di attivazioni

- Ogni chiamata di metodo crea una nuova attivazione
- All'interno di un metodo posso chiamare altri metodi e quindi le attivazioni si impilano una sopra l'altra

All'interno di una attivazione

- Lo spazio di memoria all'interno di una certa attivazione è anch'esso strutturato come una pila
- Gli elementi della pila sono chiamati frame
- Ogni frame contiene delle associazioni fra nomi di variabili e valori delle stesse
- Ogni frame rappresenta un blocco di esecuzione del programma (codice tra { })

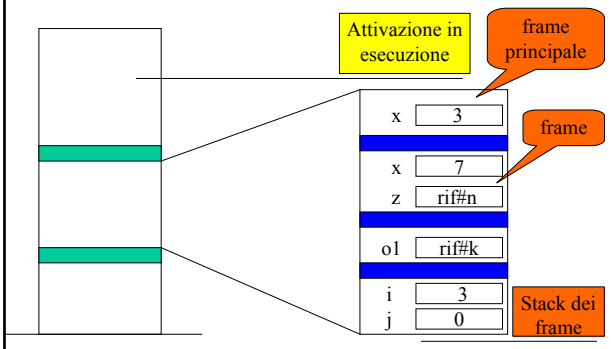
Sottoblocchi

- Il frame più in basso nella pila all'interno di una attivazione corrisponde al blocco principale dell'attivazione
- All'entrata di un sottoblocco viene creato ed impilato un altro frame
- Sottoblocchi di sottoblocchi di sottoblocchi..... => pila di frame
- I sottoblocchi gestiscono la visibilità (lo scope) delle variabili (chiariremo bene in seguito)

Visibilità

- In ogni momento l'attivazione che si trova in testa alla pila è quella in esecuzione
- Non c'è nessun rapporto tra le variabili dello stack di frame di una attivazione con quelle di una qualsiasi altra attivazione (sia sopra che sotto nella pila)
- All'interno di una attivazione lo stack di frame gestisce un certo rapporto fra le variabili di ogni frame
- Es: variabili con lo stesso nome in frame diversi: in ogni momento è visibile solo quella che si trova nel frame più in alto nella pila

Stack delle attivazioni



Variabili

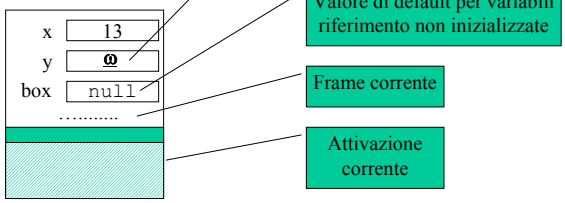
- Le variabili che si trovano all'interno dei frame possono essere di due tipi:
  1. Tipo base del linguaggio: `int`, `double`, `char`,...
  2. Tipo riferimento o Tipo oggetto
- Le variabili del tipo 1 contengono i valori caratteristici del tipo (esempio `int` interi)
- Le variabili del tipo 2 contengono **riferimenti ad oggetti** che si trovano nello heap

Dichiarazione di variabili

- *Tipo nomeVariabile*;
- *Tipo nomeVariabile* = *valore_iniziale*;
- La variabile viene creata nel frame che si trova in testa alla pila dell'attivazione corrente (quella in testa alla pila di attivazioni)
- La variabile ha come *scope* (contesto) il blocco in cui è dichiarata: al di fuori non esiste
- I parametri di un metodo, alla chiamata, sono visti come variabili che vengono create automaticamente ed inizializzate con i valori passati

Variabili

```
int x = 13;
int y;
Rectangle box;
```



Variabili

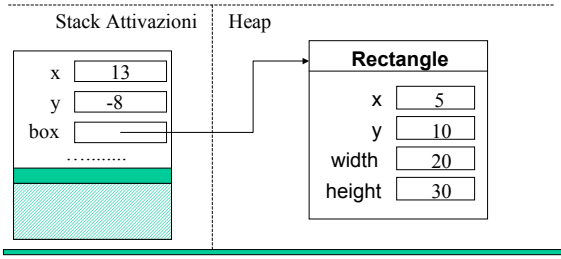
- Le variabili non inizializzate non possono essere usate fino a quando non viene assegnato loro un valore
- L'operatore = è l'assegnamento:
  - x = E;
  - Valuta l'espressione E
  - Controlla se il valore **v** ottenuto è compatibile con il tipo della variabile x
  - Se sì, copia **v** nella variabile x cancellando quello che conteneva precedentemente

Variabili riferimento

- Come tipo hanno il nome di una certa classe
- Possono contenere un riferimento ad un oggetto di quella classe (o compatibile) che risiede nello heap
- **Non contengono l'oggetto, ma il riferimento**
- Il valore lo rappresentiamo o con "rif#n" o con una freccia che punta all'oggetto nello heap

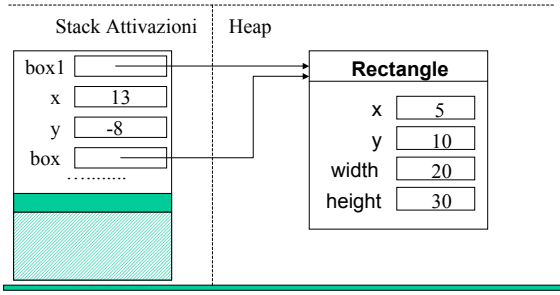
Variabili riferimento

```
box = new Rectangle(5,10,20,30);
y = -10 + 2;
```



Variabili riferimento

```
Rectangle box1 = box;
```



Codice: MoveTest.java

```
import java.awt.Rectangle;
public class MoveTest {
  public static void main(String[] args) {
    Rectangle box = new Rectangle(5,10,20,30);
    // sposta il rettangolo di un vettore
    box.translate(15,25);
    // stampa il rettangolo spostato
    System.out.println(box);
  }
}
```

- Stampa: `java.awt.Rectangle[x=20,y=35,width=20,height=30]`

Importazione di classi

- Le classi fornite con la SDK sono raggruppate in pacchetti (packages)
- Ogni pacchetto contiene classi affini
- `java.awt` è un pacchetto che contiene classi utili per disegnare finestre e forme grafiche (awt = Abstract Windowing Toolkit)
- `import` importa una certa classe come in `import java.awt.Rectangle;`
- Oppure un intero pacchetto: `import java.awt.*;`

Importazioni di classi

- Qualunque classe che non abbiamo definito, come classe pubblica, nella stessa cartella in cui si trova il file sorgente che stiamo scrivendo risulterà sconosciuta al compilatore
- A meno che:
  - La importiamo (o importiamo tutto il pacchetto che la contiene)
  - Fa parte del pacchetto `java.lang` o `java.io` che viene importato automaticamente
- Es: `String` fa parte di `java.lang`

Stampare un oggetto

- La classe `PrintStream` ha anche un metodo `println` per stampare oggetti:

```
System.out.println(new Rectangle(5,10,20,30));
```

- Stampa:

```
java.awt.Rectangle[x=5,y=10,width=20,height=30]
```

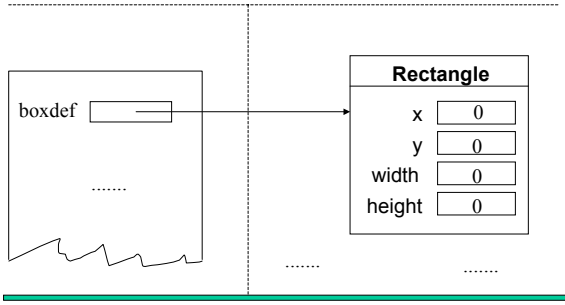
- La stampa è una descrizione testuale dei valori dei campi dell'oggetto in questione
- In questo esempio l'oggetto viene creato, stampato e poi, non avendo salvato il suo riferimento, "cancellato"

Tanti costruttori

- La stessa classe può avere più costruttori, a seconda di quanti modi ci possono essere per inizializzare i valori dei campi
- Ogni costruttore si diversifica in base ai tipi e alla quantità dei suoi parametri
- Per ogni classe esiste un costruttore di default che non prende nessun parametro ed assegna alle variabili istanza dell'oggetto valori di default

Costruttore di default

```
Rectangle boxdef = new Rectangle();
```



Valori di default

- Per variabili numeriche (`int`, `float`, `double`, `byte`): 0 o 0.0
- Per caratteri (`char`): carattere nullo a valore numerico -1
- Per variabili riferimento: `null`
- Questi valori di default valgono solo per le variabili istanza di un oggetto, non per le variabili dei frame
- Se le variabili di frame non sono inizializzate il compilatore genera **sempre** errore se vengono usate prima di essere assegnate (per convenzione, valgono **ω**, i tipi base, e `null` le variabili riferimento)