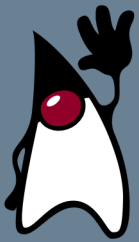




UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Laboratorio di Programmazione ad Oggetti

Ph.D. Juri Di Rocco
juri.dirocco@univaq.it
<https://jdirocco.github.io/>





Sommario

- › Code convention
- › Array
- › Argomenti a riga di comando
- › Operatori e precedenza
- › Conversione e promozione



Code Convention (1)

- › Java ha una convenzione per la definizione dei package, classi, interfacce, costanti, ...
- › Vi sono anche delle convenzioni per l'indentazione del codice
- › Tali convenzioni aiutano a rendere il codice più leggibile e ad eliminare alcuni conflitti di nomi
- › Si **raccomanda** di utilizzare le convenzioni adottate
- › Link ufficiale
 - <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>



Code Convention (2)

› Nomi classi ed interfacce

- Si utilizzano tipicamente nomi o frasi non eccessivamente lunghe
- Se è una frase composta nomi singoli scritti in maiuscolo
- Nome inizia sempre in maiuscolo
- Tipicamente le interfacce sono **aggettivate**

– Esempi

- › `ClassLoader`
- › `SecurityManager`
- › `Thread`
- › `BufferedInputStream`
- › `Runnable`
- › `Cloneable`



Code Convention (3)

› Nomi dei metodi

- Si utilizzano tipicamente verbi o frasi
- Nome inizia sempre in minuscolo
- Se è una frase composta nomi singoli scritti in maiuscolo
- Esempi
 - › `getPriority, setPriority`
 - › `length`
 - › `toString`
- Se restituisce un booleano tipicamente rispetto ad una condizione V si utilizza `isV`
 - › `isInterrupted` classe `Thread`



Code Convention (4)

› Nomi dei campi

- Generalmente identificano nomi o frasi oppure abbreviazioni
- Se non sono `final` iniziano con lettera minuscola
- Se sono composti le altre parole lettera maiuscola
- Esempi
 - › `buf, pos, count` di `java.io.ByteArrayInputStream`
 - › `out` classe `System`
- Nel caso di **costanti** (ovvero **final** e campi nelle interfacce) i nomi sono scritti in maiuscolo e se composti singole parole separati da `_`
 - › `MIN_VALUE, MAX_VALUE`

```
interface ProcessStates {  
    int PS_RUNNING = 0;  
    int PS_SUSPENDED = 1;  
}
```



Code Convention (5)

- › Variabili locali e nomi parametri metodi
 - Generalmente sono nomi brevi
 - Possono identificare
 - › Degli acronimi
 - `cp` variabile locale per classe `ColoredPoint`
 - › Abbreviazioni
 - `buf` per un buffer di qualche tipo
 - › Termini mnemonici
 - `off` e `len` oppure parametri `read` e `write` nei nomi dei metodi dell'interfaccia `DataInput` e `DataOutput`



Code Convention (6)

›

– Nel caso di variabili locali si possono usare le seguenti convenzioni

- › b per i tipi byte
- › c per i tipi char
- › d per i tipi double
- › e per i tipi Exception
- › f per i tipi float
- › i, j, k per i tipi int
- › l per i tipi long
- › o per i tipi Object
- › s per i tipi String



Code Convention (7)

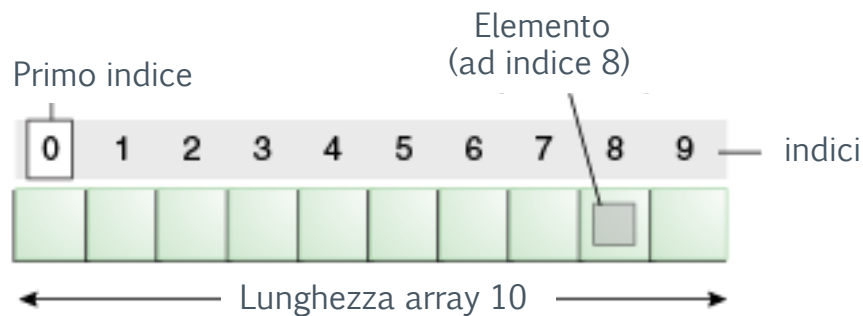
› Nomi dei package

- Tipicamente si utilizza un prefisso che identifica il nome del dominio internet (utilizzato al contrario) della società/ente che sviluppa l'applicazione
- Generalmente si fa seguire tale prefisso con il nome dell'applicazione o del progetto o del dipartimento, ...
- Esempio
 - › `it.univaq.disim.oop`
 - › `org.apache.xml.axis`
 - › `org.springframework`
 - › `javax.persistence.api`



Array

- › Struttura contenente valori dello stesso tipo
- › Si accede a ciascun valore dell'array mediante un *indice* intero
- › La lunghezza viene stabilita all'atto della sua creazione
- › Dopo la creazione la lunghezza **non** può essere variata





Dichiarazione

- › Possibile dichiarare array di tipo primitivo e di tipo reference (classe, interfacce)
- › Due modi per dichiarare un array
 - `char[] s;`
 - `Point[] p;`

 - `char s[];`
 - `Point p[];`
 - `Point p1, p2[];`

 - `Point[][] p; //Array multidimensionali`



Creazione ed accesso

› Creazione

- Un array è un oggetto quindi viene creato con `new`
- Viene creato dello spazio in memoria per contenere il riferimento e i riferimenti dell'array
- Non vengono creati gli oggetti!

```
alfabeto = new char[2];
```

```
points = new Point[10];
```

› Accesso

- `anArray[i]`

› Lunghezza

- Proprietà `length` ritorna la lunghezza dell'array

```
alfabeto = new char[2];
```

```
alfabeto.length ritorna 2
```



Cambiare la dimensione di un array in java

- › Creare un nuovo array delle dimensioni desiderate e copiare tutti gli elementi dall'array originale a quello appena creato usando:

```
java.lang.System.arraycopy(...);
```


- › Use `java.util.Arrays.copyOf(...)` methods which returns a bigger array, with the

- › ...

- › Usare un implementazione **dell'interfaccia** `Collection` (ad `java.util.ArrayList<T>`), che incrementa la dimensione dell'array quando hai bisogno.



Inizializzazione

Array anonimo 

```
boolean[] answers = { true, false, true, true, false };
```

```
Point[] points = {new Point(0,1), new Point(1,2),  
                  new Point(2,3)};
```

```
Point[] points;  
points = new Point[3];  
points[0] = new Point(0,0);  
points[1] = new Point(1,1);  
points[2] = new Point(2,2);
```



Array Multidimensionali

› Esempi

```
int[][] a = new int [4][];
```

```
a[0] = new int[5];
```

```
a[1] = new int[5];
```

```
int[][] a = new int [][4]; //POSSO?
```

```
int[][] a = new int[4][5]; //POSSO?
```



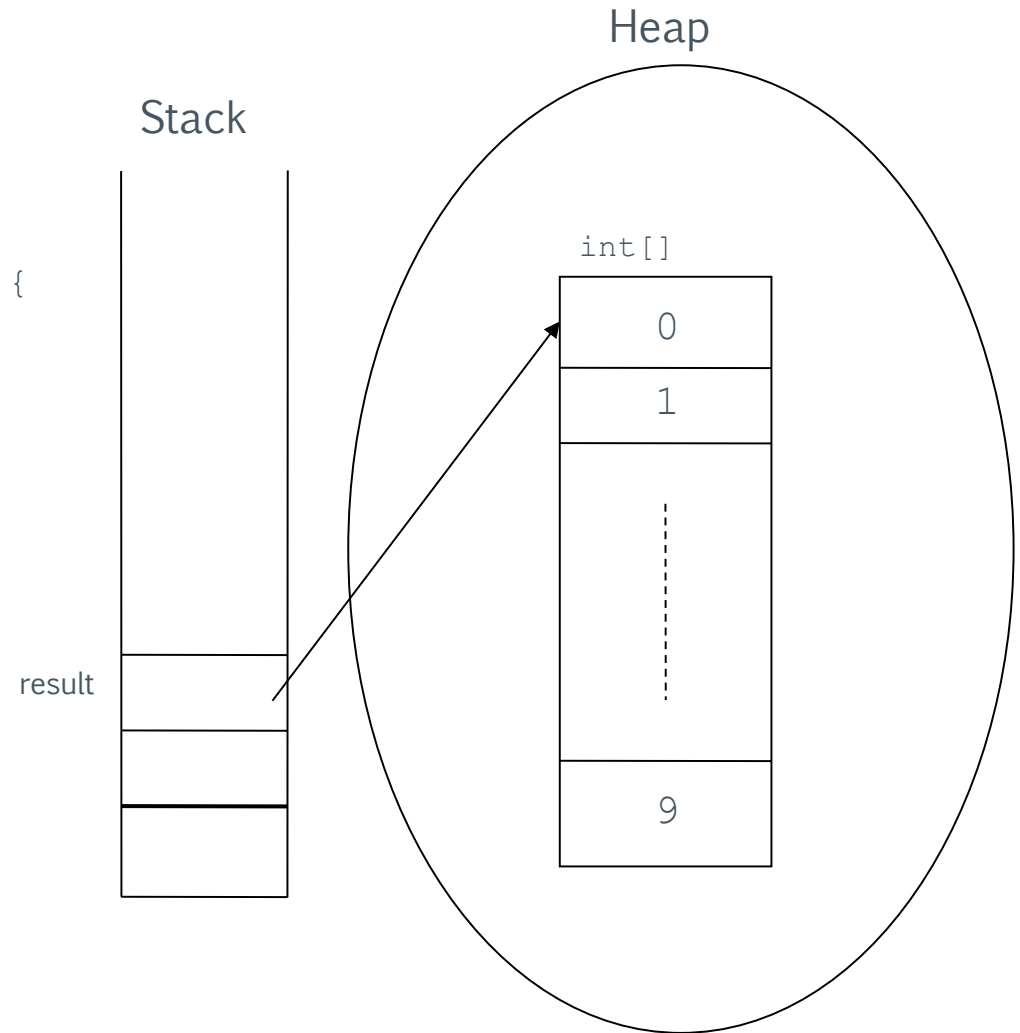
Array Multidimensional

```
public class ArrayOfArraysDemo {  
    public static void main(String[] args) {  
        String[][] cartoons = { { "Flintstones", "Fred", "Wilma", "Pebbles", "Dino" },  
                                  { "Rubbles", "Barney", "Betty", "Bam Bam" },  
                                  { "Jetsons", "George", "Jane", "Elroy", "Judy", "Rosie", "Astro"},  
                                  { "Scooby Doo Gang", "Scooby Doo", "Shaggy", "Velma", "Fred", "Daphne" } };  
        for (int i = 0; i < cartoons.length; i++)  
            System.out.print(cartoons[i][0] + ": ");  
        for (int j = 0; j < cartoons[i].length; j++) {  
            System.out.print(cartoons[i][j] + " ");  
        }  
        System.out.println();  
    }  
}
```




Rappresentazione dati (1): Array

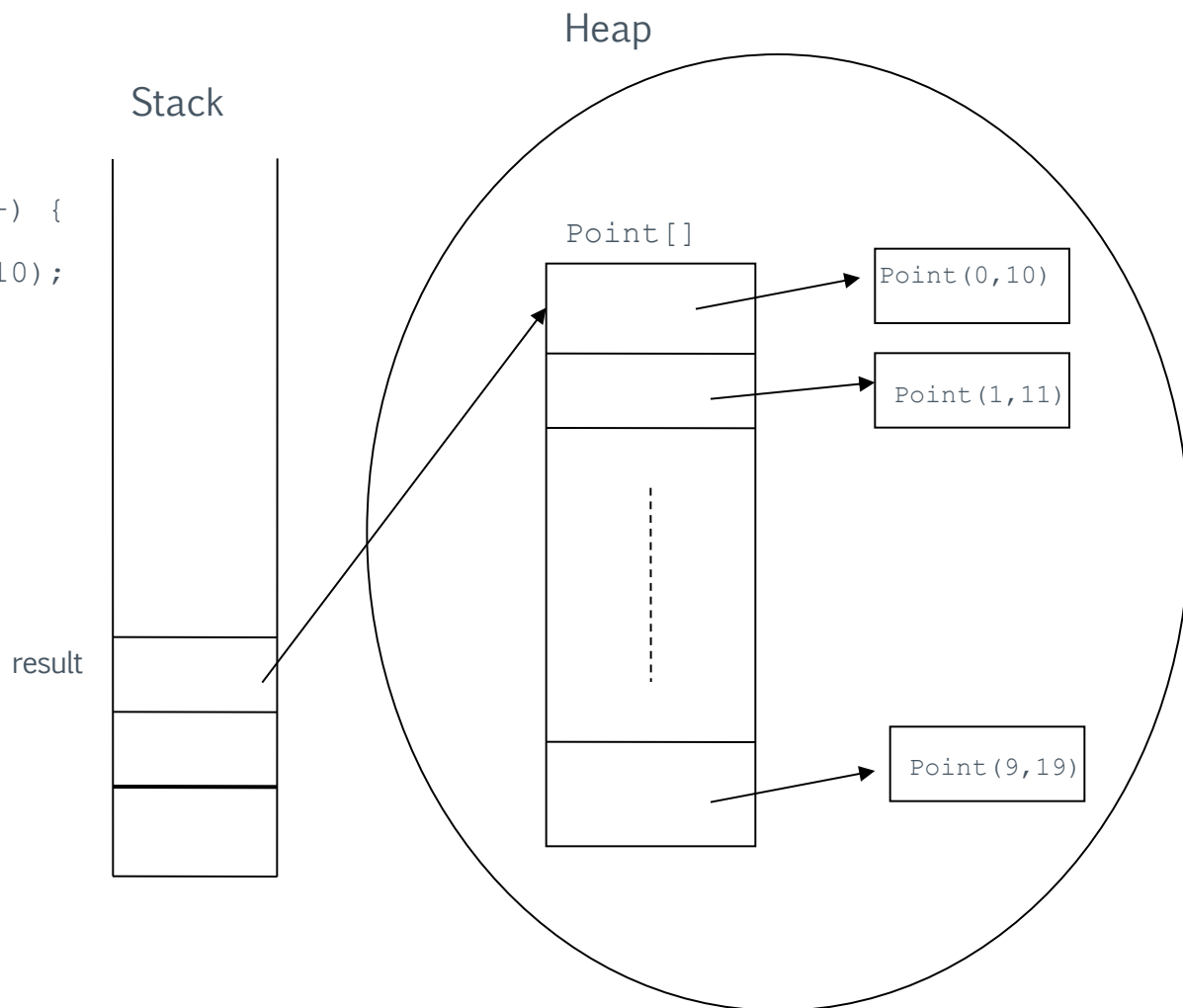
```
...  
public int[] getArray() {  
    int[] result;  
    result = new int[ 10 ];  
    for(int i=0; i<result.length;i++) {  
        result[i] = i;  
    }  
    return result;  
}
```





Rappresentazione dati (2): Array

```
public Point[] getArray() {  
    Point[] result;  
    result = new Point[ 10 ];  
    for( int i=0; i<result.length;i++) {  
        result[i] = new Point(i,i+10);  
    }  
    return result;  
}
```





Argomenti a riga di comando (1)

- › E' possibile passare ad un'applicazione Java argomenti cosiddetti *command-line* (riga di comando)
- › Costituiscono delle stringhe che vengono poste dopo il nome della classe
 - `java Test arg1 arg2 "argomento con spazio"`
- › Gli argomenti vengono memorizzati all'interno dell'array che viene passato al `main`
 - `public static void main(String[] args)`



Argomenti a riga di comando (2)

```
public class Test {  
    public static void main(String[] args) {  
  
        for ( int i = 0; i < args.length; i++ ) {  
            System.out.println("args[" + i + "]= " + args[i]);  
        }  
  
    }  
  
}
```

```
java Test arg1 arg2 "argomento con spazio"
```



Operatori

- › E' una funzione che agisce su uno, due o tre operandi
- › Operatori che richiedono un operando sono detti unari (es.: $i++$)
- › Operatori che richiedono due operando sono detto binari (es.: $a + b$)
- › Operatori che richiedono tre operandi sono detti ternari: $? :$ (forma particolare di if-else)
- › Tipi
 - Aritmetici
 - Relazionali e condizionali
 - Shift e logici
 - Assegnamento
 - Altri



Operatori Aritmetici (1)

Operatore	Uso	Descrizione
+	op1 + op2	Addizione
-	op1 - op2	Sottrazione
*	op1 * op2	Moltiplicazione
/	op1 / op2	Divisione
%	op1 % op2	Calcola il resto



Operatori Aritmetici (2)

- › Divisione tra interi restituisce un intero ovvero viene troncato
 - Esempio: $15 / 2 = 7$
- › Altrimenti divisione tra numeri in virgola mobile
 - Esempio: $15.0 / 2 = 7.5$
- › Divisione intera per zero genera eccezione (`ArithmeticException`)
- › Divisione virgola mobile da come risultato infinito ovvero `Double.POSITIVE_INFINITY` (o negativo)



Operatori Aritmetici (3)

Operatore	Uso	Descrizione
++	op++	Post-incremento
--	op--	Post-decremento
++	++op	Pre-incremento
--	--op	Pre-decremento



Operatori Relazionali (1)

- › Generano sempre un risultato di tipo `boolean`
- › Applicato ai tipi primitivi tranne `==` e `!=` che può essere applicato a tutti gli oggetti
 - Nel caso di oggetti viene confrontato il valore del **reference** della variabile
- › Esempi
 - `3 > 5`
 - `a == b` (`a` e `b` sono di tipo `Point`)



Operatori Relazionali (2)

Operatore	Uso	Descrizione
>	<code>op1 > op2</code>	op1 è maggiore di op2
>=	<code>op1 >= op2</code>	op1 è maggiore o uguale di op2
<	<code>op1 < op2</code>	op1 è minore di op2
<=	<code>op1 <= op2</code>	op1 è minore o uguale di op2
==	<code>op1 == op2</code>	op1 è uguale di op2
!=	<code>op1 != op2</code>	op1 è diverso di op2



Operatori Condizionali (1)

Operatore	Uso	Descrizione
&& (short-circuit)	op1 && op2	true se op1 e op2 sono true. op2 viene valutata se op1 è true
(short-circuit)	op1 op2	true se op1 oppure op2 sono true. op2 viene valutata se op1 è false
!	!op1	true se op1 è false. false se op1 è true.
&	op1 & op2	true se op1 e op2 sono true. op2 viene comunque valutata
	op1 op2	true se op1 oppure op2 sono true. op2 viene comunque valutata
^	op1 ^ op2	true se op1 oppure op2 sono true ma non ambedue



Operatori Condizionali (2)

- › Generano sempre un risultato di tipo `boolean`
- › Gli argomenti sono di tipo `boolean`
- › Esempio

```
String s = null;  
if ( (s!=null) & (s.length()!=10))  
    System.out.println(s + "World");  
else  
    System.out.println("Juri");
```

```
String s = null;  
if ( (s!=null) && (s.length()!=10))  
    System.out.println(s + "World");  
else  
    System.out.println("Juri");
```



Operatori Bitwise (1)

- › Vengono applicati a numeri di tipo intero
- › Permettono l'accesso diretto alla rappresentazione binaria dei dati
- › Effettuano operazioni dell'algebra booleana sulle coppie di bit corrispondenti nei due argomenti
- › Esempi

`int a = 100` `=` `1100100`

`int b = 70` `=` `1000110`

`a & b` `=` `1000100`

`a | b` `=` `1100110`



Operatori Bitwise (2)

Operatore	Uso	Descrizione
&	<code>op1 & op2</code>	Effettua un AND bit a bit tra op1 e op2
	<code>op1 op2</code>	Effettua un OR bit a bit tra op1 e op2
~	<code>~op1</code>	Effettua un NOT bit a bit di op1
^	<code>op1 ^ op2</code>	Effettua uno XOR bit a bit tra op1 e op2



Operatori Scorrimento (1)

Operatore	Uso	Descrizione
>>	op1 >> op2	Effettua uno scorrimento a dx di op2 bit su op1. I bit «vuoti» (quelli piu' a sx) vengono riempiti con lo stesso valore del bit del segno
<<	op1 << op2	Effettua uno scorrimento a sx di op2 bit su op1. I bit «vuoti» (quelli piu' a dx) vengono riempiti con lo 0
>>>	op1 >>> op2	Effettua uno scorrimento a dx di op2 bit su op1 senza considerare il bit del segno. I bit «vuoti» (quelli piu' a sx) vengono riempiti con lo 0



› Esempio

[illegible]

```
int b = -256;
```

[illegible]

```
a << 4 = 1024 // 00000000000000000000000100000000
```

$$b \ll 2 = -1024$$
$$b \gg 2 = -64$$

```
int a = -1 // 111111111111111111111111111111111111111111111111111 = -1
```

```
a >>> 24 // 00000000000000000000000001111111 = 255
```




Precedenza (1)

- › Stabilisce come viene calcolata un'espressione in presenza di diversi operatori
- › Gli operatori hanno una precedenza implicita ben definita che determina l'ordine con cui vengono valutati
 - Moltiplicazione, divisione e resto sono valutati prima di somma, sottrazione e concatenazione tra stringhe
- › Gli operatori che hanno la stessa precedenza sono valutati da sinistra a destra
- › Mediante le parentesi si può alterare l'ordine di precedenza



Operatore Assegnamento (1)

- › = è considerato un operatore
- › Prende il valore che si trova alla destra (r-value) e lo copia nell'entità di sinistra (l-value)
- › l-value deve essere il nome di una variabile
 - `a = 4` //OK
 - `4 = a` //NOT OK
- › Nel caso di tipi primitivi della variabile viene copiato il valore
- › Tipi reference viene copiato il riferimento dell'oggetto

```
int a = 10;  
int x , y;  
x = y = a;  
System.out.println(x);  
System.out.println(y);
```



Operatore Assegnamento (2)

Operatore	Uso	Equivalente a
<code>+=</code>	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>a %= b</code>	<code>a = a % b</code>
<code>&=</code>	<code>a &= b</code>	<code>a = a & b</code>
<code> =</code>	<code>a = b</code>	<code>a = a b</code>
<code>^=</code>	<code>a ^= b</code>	<code>a = a ^ b</code>
<code><<=</code>	<code>a <<= b</code>	<code>a = a << b</code>
<code>>>=</code>	<code>a >>= b</code>	<code>a = a >> b</code>
<code>>>>=</code>	<code>a >>>= b</code>	<code>a = a >>> b</code>



Precedenza (2)

	Associatività	Operatori
Più Alta	Dx a Sx	++ -- + - ~ ! (tipo di dato)
	Sx a Dx	* / %
	Sx a Dx	+ -
	Sx a Dx	<< >> >>>
	Sx a Dx	< > <= >= instanceof
	Sx a Dx	== !=
	Sx a Dx	&
	Sx a Dx	^
	Sx a Dx	
	Sx a Dx	&&
	Sx a Dx	
	Sx a Dx	?: OPERATORE TERNARIO
Più Bassa	Sx a Dx	= *= /= %= += -= <<= >>= >>>= &= ^= =



Conversione e Promozione (1)

- › Ogni **espressione** in Java ha un tipo che può essere dedotto dalla struttura dell'espressione, dal tipo dei letterali, variabili e metodi menzionati nell'espressione
- › E' possibile utilizzare un'espressione in un contesto dove il tipo non è appropriato
- › In alcuni casi porta ad errore (es. `if` può avere soltanto espressioni con tipo `boolean`)
- › Altri casi il contesto accetta in modo implicito l'espressione anche di tipo diverso (conversione)



Conversione (1)

- › Categorie di conversione
 - Identity
 - Widening Primitive
 - Narrowing Primitive
 - Widening e Narrowing Reference
 - String
 -



Conversione Identity

- › E' permessa la conversione da un tipo allo stesso tipo
- › E' ovvia ma permette di introdurre cast superflui per aumentare la chiarezza del codice
- › Espressione tipo `boolean` ammette soltanto conversione Identity da `boolean` a `boolean`
 - `int i = (5 <= 3); //Non Ammessa`



Conversione Widening Primitive (1)

- › Conversioni sui tipi primitivi ammessi
 - byte a short, int, long, float, o double
 - short a int, long, float, o double
 - char a int, long, float, o double
 - int a long, float, o double
 - long a float o double
 - float a double
- › In generale, **Non vi è perdita di informazioni** ovvero il valore numerico è conservato
- › Conversione da int o long a float oppure da long a double potrebbe comportare perdita di precisione ovvero si potrebbe avere una perdita dei bit più significativi
- › Non vengono mai generate delle eccezioni



Conversione Widening Primitive (2)

```
class Test {  
    public static void main(String[] args) {  
        int big = 1234567890;  
        float approx = big;  
        System.out.println(big - (int)approx);  
    }  
}
```

- › Output: 46
- › Perdita di informazioni poiché il valore `float` non è preciso alle dieci cifre



Conversione Narrowing Primitive (1)

› Conversioni sui tipi primitivi ammessi

- short **a** byte **o** char
- char **a** byte **o** short
- int **a** byte, short, **o** char
- long **a** byte, short, char, **o** int
- float **a** byte, short, char, int, **o** long
- double **a** byte, short, char, int, long, **o** float

› **Vi è perdita di informazione**

- › Vengono presi i bit meno significativi relativi al tipo destinatario (quindi vi può essere il cambio di segno)
- › Non vengono mai generate delle eccezioni



Conversione Narrowing Primitive (2)

```
class Test {  
    public static void main(String[] args) {  
        float fmin = Float.NEGATIVE_INFINITY;  
        float fmax = Float.POSITIVE_INFINITY;  
  
        System.out.println("long: " + (long)fmin + ".." + (long)fmax);  
        System.out.println("int: " + (int)fmin + ".." + (int)fmax);  
        System.out.println("short: " + (short)fmin + ".." + (short)fmax);  
        System.out.println("char: " + (int)(char)fmin + ".." + (int)(char)fmax);  
        System.out.println("byte: " + (byte)fmin + ".." + (byte)fmax);  
    }  
}
```

Output

```
long: -9223372036854775808..9223372036854775807  
int: -2147483648..2147483647  
short: 0..-1  
char: 0..65535  
byte: 0..-1
```



Conversione String

- › E' possibile convertire qualsiasi tipo al tipo `String` incluso il tipo `null`



Conversioni non ammesse

- › Da un tipo reference a un tipo primitivo e viceversa (eccetto per la conversione String)
- › Da `null` a un tipo primitivo
- › Tipo classe `S` a una classe di tipo `T` se `S` non è sottoclasse di `T` e viceversa
- ›
- ›
- ›



Contesti di Conversione

- › Le espressioni si possono trovare in 5 *contesti di conversione*
 - Assegnamento
 - Invocazione del metodo
 - Casting
 - `String`
 - Promozione numerica
 - › Si ha nell'espressioni dove compaiono operatori numerici (Es.: +, *)
 - › Si parla di promozioni poiché la conversione può dipendere in parte dal tipo dell'altro operando nell'espressione
- › Ogni contesto permette conversione in una delle categorie precedenti



Contesto conversione Assegnamento (1)

- › Si verifica quando il valore di un'espressione è assegnato ad una variabile → il tipo dell'espressione è convertito al tipo della variabile
- › Conversioni ammesse
 - Identity
 - Widening primitive e reference
 - Narrowing primitive se tutte le seguenti condizioni sono verificate
 - › Espressione è una costante di tipo `byte` `short` `char` o `int`
 - › Tipo della variabile è `byte` `short` o `char`
 - › Valore dell'espressione (conosciuto a tempo di compilazione) è rappresentabile all'interno del tipo della variabile
- › Si verifica un errore in **compilazione** se non è possibile convertire il tipo dell'espressione al tipo della variabile
- › Esempio
 - `byte theAnswer = 42; //OK per ultimo punto`
 - `short s1 = 0x1ffff; // COSA SUCCEDDE QUI?`
 - `short s2 = 0x1ffff; // COSA SUCCEDDE QUI?`

 - `byte b = 50;`
 `b = b * 2; //Errore in compilazione`



Contesto conversione Assegnamento (2)

```
class Test {  
    public static void main(String[] args) {  
        short s = 12;                // narrow 12 to short  
        float f = s;                 // widen short to float  
        System.out.println("f=" + f);  
        char c = '\u0123';  
        long l = c;                  // widen char to long  
        System.out.println("l=0x" + Long.toString(l,16));  
        f = 1.23f;  
        double d = f;                // widen float to double  
        System.out.println("d=" + d);  
    }  
}
```

Output

f=12.0

l=0x123

d=1.2300000190734863



Contesto conversione Assegnamento (3)

```
class Test {  
    public static void main(String[] args) {  
        short s = 123;  
        char c = s;          // error: would require cast  
        s = c;               // error: would require cast  
        c = 65535;           //OK  
        c = 65536;           // error  
    }  
}
```



Contesto invocazione di un metodo (1)

- › E' applicata ad ogni valore di un parametro in una invocazione di un metodo o di un costruttore
- › Tipo dell'espressione deve essere convertito al corrispondente tipo del parametro
- › Conversioni ammesse
 - Identity
 - Widening primitive e reference
- › Non vengono incluse conversioni narrowing di costanti intere



Contesto invocazione di un metodo (2)

```
class Test {  
    static int m(byte a, byte b) { return a+b; }  
    static int m(short a, short b) { return a-b; }  
  
    public static void main(String[] args) {  
        System.out.println(m(12, 2));  
    }  
}
```

Errore in compilazione poiché 12 e 2 sono letterali di tipo `int`



Contesto String

- › Si applica soltanto agli operandi dell'operatore binario + dove uno degli argomenti è una stringa
- › Esempio

```
System.out.println(12 + " Hello");
```



Contesto Casting (1)

- › Viene applicato all'operando di cast ()
- › Tipo dell'espressione viene convertito al tipo esplicito indicato nell'operatore di cast
- › Conversioni ammesse
 - Identity
 - Widening o narrowing primitive
 - Widening o narrowing reference
- › Alcuni cast producono errore in compilazione
 - Casting tra tipo primitivo e reference e viceversa
- › E' ammesso il casting tra qualsiasi tipo primitivo
- › Esempio di casting tra tipo classe S a un tipo classe T
 - S e T devono essere in relazione ovvero stessa classe, oppure S sotto-classe di T o viceversa altrimenti si verifica un errore in compilazione
- › Altri casi presenti (vedere libro della specifica)



Contesto Casting (2)

```
class Test {  
    static int m(byte a, byte b) { return a+b; }  
    static int m(short a, short b) { return a-b; }  
  
    public static void main(String[] args) {  
        byte b = 10;  
        b = (byte) (b * 2);  
        System.out.println(m((byte)12, (byte)2));  
    }  
}
```



Contesto Casting (3)

```
class Point {  
    int x, y;  
}
```

```
class ColoredPoint extends Point {  
    int color;  
    public void setColor(int c) {  
        color = c;  
    }  
}
```



Contesto Casting (4)

```
class Test {  
    public static void main(String[] args) {  
        Point p = new Point();  
        ColoredPoint cp = new ColoredPoint();  
        cp = (ColoredPoint)p;           //Errore a run-time  
        Long l = (Long)p;               //Errore in compilazione  
        int i = (int) p;                //Errore in compilazione  
    }  
}
```




Contesto promozione numerica

- › Applicata agli operandi di un operatore aritmetico
- › Conversioni ammesse
 - Identity
 - Widening primitive
- › Viene utilizzata per convertire gli operandi dell'operatore ad un tipo comune prima di eseguire l'operazione
- › Tipi di promozione
 - Unaria
 - Binaria



Promozione numerica Unaria (1)

- › Applicata agli operatori con un singolo operando
- › Se l'operando è di tipo `byte`, `short` o `char` → l'operando viene promosso ad `int`
- › Altrimenti non viene convertito
- › Applicata alle seguenti espressioni
 - Dimensione nella creazione di un array
 - Indice in un accesso ad un array
 - Operando dell'operatore unario `+` e `-`
 - Operando dell'operatore complemento `~`
 - Ogni singolo operando degli operatori di shift `>>`, `>>>` e `<<`



Promozione numerica Unaria (2)

```
class Test {  
    public static void main(String[] args) {  
        byte b = 2;  
        int a[] = new int[b];          //Promozione  
        char c = '\u0001';  
        a[c] = 1;                      //Promozione  
        a[0] = -c;                     //Promozione  
        System.out.println("a: " + a[0] + ", " + a[1]);  
        b = -1;  
        int i = ~b;    //Promozione  
        System.out.println("~0x" + Integer.toHexString(b) + " == 0x" + Integer.toHexString(i));  
        i = b << 4L;  //Promozione operando di sinistra  
        System.out.println("0x" + Integer.toHexString(b) + "<<4L == 0x" + Integer.toHexString(i));  
    }  
}  
  
    Output  
    a: -1,1  
    ~0xffffffff == 0x0  
    0xffffffff << 4L == 0xffffffff0
```



Promozione numerica Binaria (1)

- › Applicata ad una coppia di operandi dove ognuno denota un valore numerico
 - Se uno dei due operandi è `double` l'altro è convertito a `double`
 - Altrimenti se uno dei due operandi è `float` l'altro è convertito a `float`
 - Altrimenti se uno dei due operandi è `long` l'altro è convertito a `long`
 - Altrimenti entrambi gli operandi sono convertiti a `int`
- › Operatori dove si può applicare la conversione
 - `*`, `/` e `%`
 - `+` e `-`
 - `<`, `<=`, `>`, e `>=`
 - `==` e `!=`
 - Operatori bit a bit `&`, `^`, e `|`



Promozione numerica Binaria (2)

```
class Test {  
    public static void main(String[] args) {  
        int i = 0;  
        float f = 1.0f;  
        double d = 2.0;  
        // First int*float is promoted to float*float, then  
        // float==double is promoted to double==double:  
        if (i * f == d) System.out.println("oops");  
        // A char&byte is promoted to int&int:  
        byte b = 0x1f;  
        char c = 'G';  
        int control = c & b;  
        System.out.println(Integer.toHexString(control));  
    }  
}
```

Output

7

0.25