Tipi di dati fondamentali

Tipi primitivi: interi

- Java fornisce otto tipi primitivi indipendenti dall'implementazione e dalla piattaforma
- Interi
  - Tipo **byte** **(8 bit)**
    - Interi con segno tra -128 e 127, valore di default 0, 8 bit
  - Tipo **short** **(16 bit)**
    - Interi con segno tra -32768 e 32767, valore di default 0,
  - Tipo **int** **(32 bit)**
    - Interi con segno tra -2^{31} e 2^{31} -1, valore di default 0
  - Tipo **long** (64 bit)
    - Interi con segno tra -2^{63} e 2^{63} -1, valore di default 0

Tipi primitivi: costanti intere

- Una costante intera per default è di tipo int
- Costanti intere possono essere espresse anche in ottale (prefisso 0) o in esadecimale (prefisso 0x)
- Per costanti intere di tipo long aggiungere il suffisso **L** oppure **l**
  - Es. 4000L
- Costanti intere di tipo byte (risp. short)
  - costanti di tipo int il cui valore rientra nel range del tipo byte (risp. short)

Tipi primitivi: numeri con virgola

- Seguono standard IEEE 754
- Tipo **float** (32 bit)
  - numeri in virgola mobile con 7 cifre significative
  - compresi tra 1.4E-45 e 3.4028235E+38
  - valore di default 0.0
  - le costanti vanno terminate con **F** o **f** (es. float a=3.456F;)
- Tipo **double** (64 bit)
  - numeri in virgola mobile in doppia precisione (15 cifre significative)
  - compresi tra 4.9E-324 e 1.7976931348623157E+308
  - valore di default 0.0
  - le costanti con virgola sono di tipo double per default
    - possono essere terminate con **D** o **d** ma non è necessario

Tipi Primitivi: caratteri

- Seguono la codifica Unicode che estende ASCII su 16 bit
- Tipo **char** (16 bit)
  - Si possono usare i caratteri o i relativi codici numerici preceduti da **\u**, sempre tra apici. Es.
    - char a='A';
    - char a='\u0041' (in esadecimale su 4 cifre);
    - char a=65;
  - Caratteri con codici tra 0 e 65535
  - valore di default '\u0000' ('\0' del C)
  - Possibilità di usare \ per caratteri particolari ('\n', '\t', '\'', '\b', '\0', …)

Tipi Primitivi: boolean

- Tipo **boolean** (1 bit)
- Ammette solo due possibili valori (**true**, **false**)
- Valore di default false
- Non si possono assegnare interi alle variabili booleane
  - **false non è 0!!!**

Operatori per i Tipi Primitivi (1)

- Java ha gli stessi operatori del C, con qualche leggera differenza
- Aritmetici (**+**, **-**, ***, /**, **%**, **++**, **--**, **+=**, **-=**, ***=**, **/=**, **%=**)
  - Non sono applicabili a variabili di tipo boolean
- Relazionali (**<**, **>**, **<=**, **>=**, **==**, **!=**)
  - Producono risultati di tipo boolean (true, false)
  - **<**, **>**, **<=**, **>=** non sono applicabili a variabili di tipo boolean

Operatori per i Tipi Primitivi (2)

- Logici (**&&**, **||**, **!**, **&**, **|**, **^**)
  - && and || non valutano espressione destra se valore della condizione può essere stabilita dall'espressione sinistra (valutazione abbreviata)
- Bit a bit (solo tipi interi e char)
  - **&** (AND) , **|** (OR), **^** (XOR), **~** (complemento bit a bit)
  - Shift **<<** , **>>** (rispetta segno operando), **>>>** (mette 0 come bit più significativo)
    - Es. x << n sposta bit di x di n posizioni a sinistra e riempe i posti lasciati liberi con 0
  - Combinati assegnamento: **&=**, **|=**, **^=**, **<<=**, **>>=**, **>>>=**
  - **~**, **<<**, **>>**, **>>>**, **<<=**, **>>=**, **>>>=** non si applicano a variabili boolean

Priorità e associatività

Operatori	Associatività
[ ] ( ) . ++(postfisso) --(postfisso)	da sinistra a destra
! ~ ++(prefisso) --(prefisso) +(unario) -(unario)	da destra a sinistra
casting new	da destra a sinistra
* / %	da sinistra a destra
+ -	da sinistra a destra
>> << >>>	da sinistra a destra
= = !=	da sinistra a destra
&	da sinistra a destra
^	da sinistra a destra
|	da sinistra a destra
&&	da sinistra a destra
||	da sinistra a destra
? :	da destra a sinistra
= += -= *= /= %= &= |= ^= <<= >>= >>>=	da destra a sinistra

Tipi delle espressioni

- Il tipo delle espressioni con operatori aritmetici o bit-a-bit su interi (ad eccezione degli shift) è int a meno che un operatore è long (e in questo caso è long)
- Per gli operatori di shift non si tiene conto del tipo dell'operando destro
- Se è presente un operando in virgola mobile il tipo è float a meno che uno degli operandi sia double (e in questo caso è double)

Conversione implicita di tipo

- Ampliamento (da più piccolo a più grande):
  - byte → short → int → long → float → double
  - char → int
- Conversione da long a float
  - possibile in quanto range di float più ampio del range di long
  - perdita di precisione (da 64 a 32 bit)
- Restringimento (da più grande a più piccolo):
  - ammesso negli assegnamenti di costanti di tipo int a tipo short, byte o char a patto che il valore della costante possa essere contenuto nel tipo di destinazione

Esempi conversioni di tipo

```
int a=1000L;
```
// **Errore**: tentativo di assegnare long a int (anche se 1000 in int ci va)
```
short s=700;
byte b=-70;
int x=s+b;
```
// **Ok**: short e byte sono tipi più piccoli; converte tutto a int come in C
```
float f=1.2;
```
// **Errore**: assegnazione di double a float (anche se 1.2 in float ci va)
```
double d=700.23;
float c=-70F;
double x=d+c;
```
// **Ok**: converte tutto a double come in C
```
float y=d+c;
```
// **Errore**: converte c a double e tenta di assegnare double a float

Virgola mobile e Interi

- I tipi decimali accettano qualunque tipo di espressione intera, con eventuale arrotondamento sulle ultime cifre significative, ma a nessun tipo di intero si possono assegnare espressioni in virgola.

Esempi

- `float f=1234567L;` // **OK**: diventa 1234567.0
- `float d=12345678;` // **OK**: diventa 1.2345678E7
- `float c=-123456782;` // **OK**: arrotondamento sulle ultime cifre
- `long x=f;` // **Errore**: tenta di assegnare float a long

Char e Interi

- Ai tipi interi **long** e **int** si possono assegnare espressioni **char** che verranno convertite nel relativo codice numerico, a **byte** e **short** non si possono assegnare **char** e a **char** non si può assegnare nessun intero.

Esempi

- `char c='B';`
- `int d=44+c;` // **OK**: c viene convertito a int e d vale 110
- `char s=d;` // **Errore**: assegna int a char (anche se 110 è un valore possibile per char)
- `char x=110;` // **Ok**: 110 è un valore possibile per un char
- `char t=-7;` // **Errore**: -7 non è un valore possibile per un char

Ancora sulle Conversioni

- Ai tipi float e double si possono sempre assegnare espressioni char mentre il contrario non è mai possibile.
- **Esempi**
  - `char c='B';`
  - `float d=44+c;` // **OK**: c e 44 vengono convertiti a float e d vale 110.0
- **NOTA** Non sono possibili conversioni di tipo da/verso boolean

La divisione intera

- `Se entrambi gli operandi sono interi allora il risultato della divisione è un intero`
  - `9 / 4` è `2` e non `2.25`!
- Se si vuole che il risultato sia un numero decimale allora almeno uno degli operndi deve essere un numero in virgola mobile
  - 9 / 4.0 è 2.25

Casting sui Tipi Primitivi

- Un cast esplicito può servire a forzare le conversioni che in java non sono permesse. La sintassi è uguale a quella del C.

- **Esempi**

```
double d=-1.8345678901234567;
float f=(float)d; // perdita di precisione
int i=(int)d; // i vale -1 (non c'è arrotondamento)
short s=-700;
char c=(char)s; // possibile, ma senza senso
boolean b=(boolean)i; // Errore: non sono permessi cast da/verso boolean
```

Variabili

- In Java le variabili possono essere dichiarate ovunque nel codice
  - `int a=20;`
  - `int n=a*10;`
- Una dichiarazione consiste in uno specificatore di accesso, una serie modificatori, un tipo e un nome
- Variabili **final**
  - Il loro valore non può essere modificato (costante)
  - Possono essere dichiarate in
    - un metodo:
      final nomeTipo nomeVar = espressione;
    - una classe:
      specificatoreDiAccesso **static final** nomeTipo nomeVar = espressione;
  - Si usano in genere nomi con caratteri maiuscoli
- **Nota:** static denota una variabile della classe, quindi non ne viene creata una copia per ogni oggetto istanziato ma tutti gli oggetti fanno riferimento alla stessa variabile

Esempio

```
public class Purse
{
    public Purse()
    {
        nickels = 0;
        dimes = 0;
        quarters = 0;
    }

    public void addNickels(int count)
    {
        nickels = nickels + count;
    }

    public void addDimes(int count)
    {
        dimes = dimes + count;
    }

    public void addQuarters(int count)
    {
        quarters = quarters + count;
    }

    public double getTotal()
    {
        return nickels * NICKEL_VALUE
        + dimes * DIME_VALUE + quarters *
                QUARTER_VALUE;
    }

    private static final double NICKEL_VALUE = 0.05;
    private static final double DIME_VALUE = 0.1;
    private static final double QUARTER_VALUE = 0.25;

    private int nickels;
    private int dimes;
    private int quarters;
}
```

Esempio

```
public class PurseTest
{
   public static void main(String[] args)
   {
      Purse myPurse = new Purse();

      myPurse.addNickels(3);
      myPurse.addDimes(1);
      myPurse.addQuarters(2);

      double totalValue = myPurse.getTotal();
      System.out.print("The total is ");
      System.out.println(totalValue);
   }
}
```

Tipi primitivi e oggetti

- Valori in Java sono oggetti o tipi primitivi
  - variabili di un tipo primitivo contengono valori
  - variabili oggetto contengono riferimenti a oggetti
- Assegnamenti
  - tra variabili di tipo primitivo viene copiato il valore
    - Es. x = y; // x e y hanno lo stesso valore ma non sono // collegate
  - tra variabili oggetto viene copiato il riferimento all'oggetto
    - Es. x = y; // x e y si riferiscono allo stesso oggetto
    - per ottenere una copia di oggetti occorre invocare il metodo clone()
      - in alternativa, si può istanziare un nuovo oggetto (con lo stesso valore delle variabili istanza)

La Classe Math

- La classe **Math** del package java.lang contiene una serie di metodi *statici* (metodi della classe) da utilizzare per calcolare funzioni matematiche sui tipi primitivi.
- In genere i metodi in **Math** lavorano su **double** e restituiscono **double**, ma questo non è un limite perchè un metodo che funziona su **double** funziona anche su tutti gli altri tipi (numerici).
- **NOTA** I metodi in **Math** non possono essere chiamati su variabili di tipo boolean

Metodi di Math (1)

- I principali metodi contenuti nella classe **Math** sono:
  - Valore assoluto (implementato anche per float, int e long)
    - double Math.abs(double x)
  - Funzioni trigonometriche
    - double Math.sin(x);
    - double Math.cos(x);
    - double Math.tan(x);
    - double Math.asin(x);
    - double Math.acos(x);
    - double Math.atan(x);

Metodi in Math (2)

- Max e Min (implementati anche per float, int e long)
  - double Math.max(double x, double y)
  - double Math.min(double x, double y)
- Potenza, esponenziale, logaritmo naturale e radice quadrata
  - double Math.pow(double x, double y)
  - double Math.exp(double x)
  - double Math.log(double x)
  - double Math.sqrt(double x)

Metodi in Math (3)

- Funzioni di arrotondamento
  - double Math.ceil(double x)
  - double Math.floor(double x)
  - long Math.round(double x)
- Costanti (definite con **final** e **static**)
  - Math.PI (pi greco)
  - Math.E (base dei logaritmi naturali)

Invocazione di metodi statici

- ClassName . MethodName (parameters)
  - Metodo statico = metodo che non opera su un particolare oggetto della classe
- Esempio: `Math.round(3.14)`

Stringhe

- Sequenza di caratteri
- Oggetti della classe String
- Immutabili
  - nessun metodo di String modifica lo stato della stringa
- Stringhe costanti: `"Carl"`
- Variabili stringhe:
  ```
  String name = "Carl";
  ```
- Lunghezza di una stringa:
  ```
  int n = name.length();
  ```

Sottostringhe

- `String greeting = "Clown";`
  Le posizioni dei caratteri di una stringa sono numerate a partire da 0
  - $_0C_1l_2o_3w_4n$
- `String sub = greeting.substring(1,4);`
  Gli argomenti indicano la posizione del primo carattere della sottostringa e quella successiva all'ultimo carattere
  - Es. la stringa sub contiene low
- Se viene omesso secondo parametro si sottintende fino a fine stringa
  - `String sub = greeting.substring(1);`
    Ora sub contiene lown

Concatenazione

- ```
  String fname = "Harry";
  String lname = "Hacker";
  String name = fname + lname;
  ```
- `name` è `"HarryHacker"`
- Se un operando di `+` è una stringa, l'altro è convertito in una stringa:
  ```
  String a = "Agent";
  String name = a + 7;
  ```
- `La stringa name` è `"Agent7"`

Conversioni tra stringhe e numeri

- Da stringhe a numeri:
  - stringa contiene un numero (Es. "19" o "19.5")
    ```
    int n = Integer.parseInt(str);
    double x = Double.parseDouble(str);
    ```
- Da numeri a stringhe:

```
String str = "" + n;
str = Integer.toString(n);
str = Double.toString(d);
```
```

Programma MakePassword.java

```
public class MakePassword
{  public static void main(String[] args)
   {  String firstName = "Harold";
      String middleName = "Joseph";
      String lastName = "Hacker";

      // estrai le iniziale

      String initials = firstName.substring(0, 1)
         + middleName.substring(0, 1)
         + lastName.substring(0, 1);

      // aggiungi l'età

      int age = 19; // età dell'utente
      String password = initials.toLowerCase() + age;

      System.out.println("Your password is " + password);
   }
}
```

Leggere l' input da console (1)

- Si usa l'oggetto System.in
  - Legge solo byte
- Una stringa però è costituita da caratteri (Unicode usa 2 byte)
  - Per ottenere un lettore di caratteri si deve trasformare **System.in** in un oggetto di tipo **InputStreamReader** *(lettore di flusso di ingresso)*

```
InputStreamReader reader = new
      InputStreamReader(System.in);
```

- Un lettore di flusso però legge singoli caratteri e non un'intera stringa
  - Per leggere una riga di ingresso devo trasformare un lettore di flusso di ingresso in un oggetto di tipo **BufferedReader**

```
BufferedReader console = new BufferedReader(reader);
```

- La classe BufferedReader ha un metodo readLine che legge una riga di ingresso

```
String input = console.readLine();
```

Programma Coins.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class Coins
{  public static void main(String[] args)
                          throws IOException
      {  final double PENNY_VALUE = 0.01;
         final double NICKEL_VALUE = 0.05;
         final double DIME_VALUE = 0.1;
         final double QUARTER_VALUE = 0.25;

         InputStreamReader reader
            = new InputStreamReader(System.in);
         BufferedReader console
            = new BufferedReader(reader);
```

```
System.out.println("Quanti penny hai?");
String input = console.readLine();
int pennies = Integer.parseInt(input);

System.out.println("Quanti nickel hai?");
input = console.readLine();
int nickels = Integer.parseInt(input);

System.out.println("Quanti dime hai?");
input = console.readLine();
int dimes = Integer.parseInt(input);

System.out.println("Quanti quarter hai?");
input = console.readLine();
int quarters = Integer.parseInt(input);
```

```
            double total = pennies * PENNY_VALUE
                  + nickels * NICKEL_VALUE
                  + dimes * DIME_VALUE
                  + quarters * QUARTER_VALUE;
                     // valore totale delle monete
            System.out.println("Total value = ” +
total);

   }  //chiude il corpo del main
} //chiude la definizione della classe
```