02
Oggetti e classi

Mirko Viroli
mirko.viroli@unibo.it

C.D.L. Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2022/2023

Outline

Goal della lezione

- Illustrare i concetti base del paradigma object-oriented
- Mostrare un primo semplice programma Java
- Fornire una panoramica di alcuni meccanismi Java

Argomenti

- Oggetti e riferimenti
- Tipi primitivi
- Classi, metodi e campi
- Accenno a package e librerie
- Stampe a video
- Primo semplice programma Java

Outline

“Everything is an object”

Riferimenti ad oggetti

- Nessun meccanismo per accedere ai dati per valore o puntatore!
- Le variabili conterranno dei riferimenti agli oggetti veri e propri, sono quindi dei nomi “locali” utilizzabili per denotare l’oggetto

```
1  // Creo un oggetto String, e lo assegno al nome s
2  // questo oggetto stringa rappresenta la sequenza
3  // vuota di caratteri
4  String s = new String();
5
6  // In generale si crea un oggetto con l'operatore <new>
7  // Accetta 0,1 o più argomenti, a seconda del tipo:
8  // valori primitivi (numeri), string literals, o oggetti
9  String s2 = new String("just a string");
10 Point2D p = new Point2D(10.5,20.3);
11 Object o = new Object();
```

Variabili, oggetti, e valori primitivi

Concetti base

- Variabile: un contenitore con nome (come in C), usabile per denotare un oggetto
- Valore primitivo: p.e. un numero, anche assegnabile ad una variabile

```
1  String s = new String("just a string");
2
3  // Un caso ad-hoc di Java, equivalente a:
4  // String s2 = new String("home");
5  String s2 = "home";
6
7  // Definisco il nome s4, ma non lo inizializzo
8  String s4;
9  // ora assegno s4, da qui in poi è utilizzabile
10 s4 = "car";
11
12 // Il nome i è assegnato al valore primitivo 5
13 int i = 5;
```

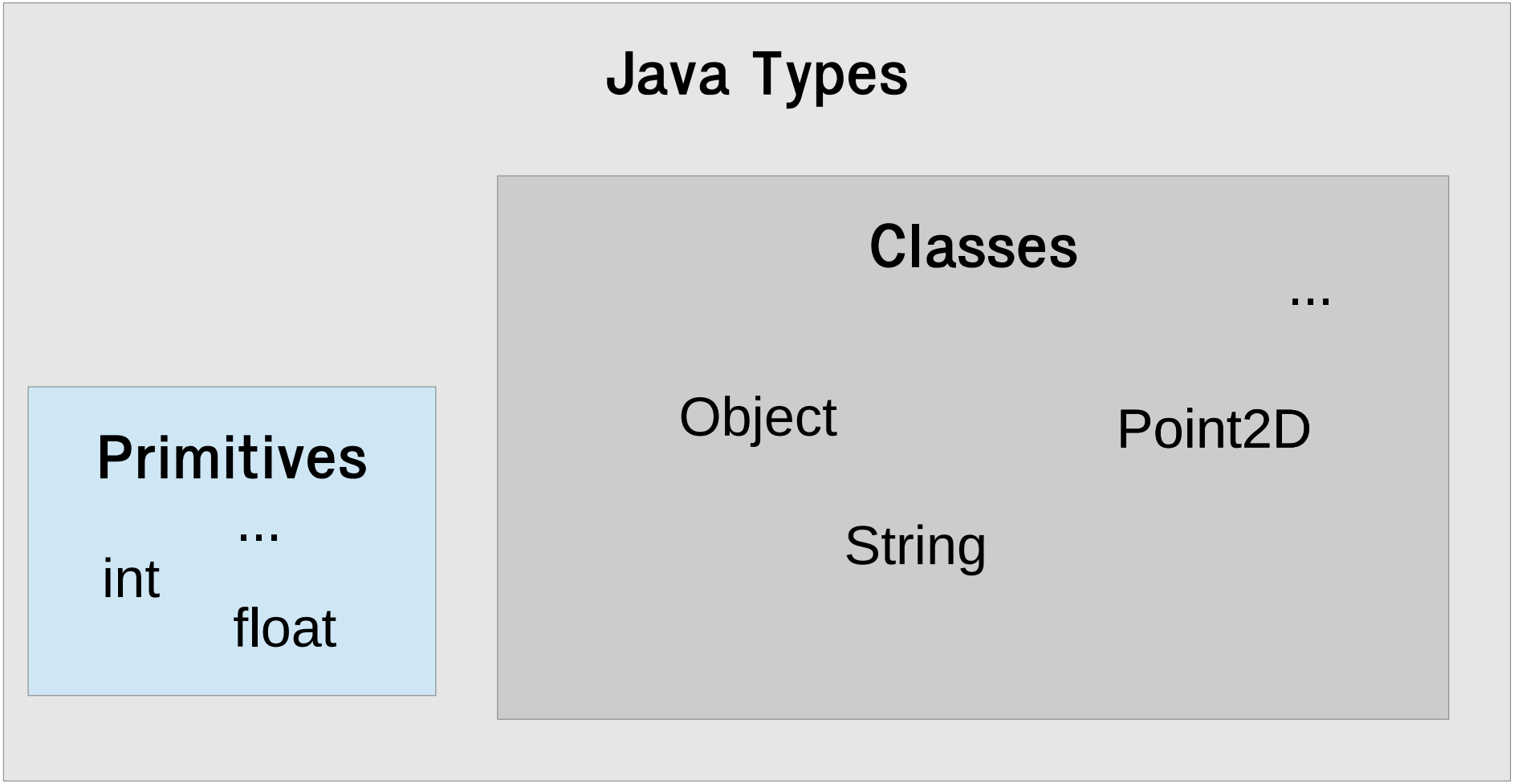
"(Almost) Everything is an object"

Tipi primitivi: tipi speciali usati per valori atomici

- Assomigliano molto a quelli del C, ma hanno dimensioni fissate
- I `boolean` possono valere `true` o `false`
- Altre classi di libreria (`BigDecimal`, `BigInteger`) gestiscono numeri di dimensione/precisione arbitraria

Primitive type	Size	Minimum	Maximum
boolean	—	—	—
char	16 bits	Unicode 0	Unicode 2^{16}- 1
byte	8 bits	-128	+127
short	16 bits	-2^{15}	$+2^{15}$-1
int	32 bits	-2^{31}	$+2^{31}$-1
long	64 bits	-2^{63}	$+2^{63}$-1
float	32 bits	IEEE754	IEEE754
double	64 bits	IEEE754	IEEE754

Una prima classificazione dei tipi



Variabili e tipi

```
1  int i = 5;                    // assegnamenti con tipi primitivi
2  double d = 145e-20;
3  boolean b = true;
4  int other = i;
5
6  Object o = new Object();      // assegnamenti con classi
7  String s = "another string";
8  Point2D p = new Point2D(10.4,20.3);
9  Point2D q = p;                // assegnamento da/a variabile
10
11 Object on = null;             // assegnamento a null
```

Costrutto `var`: "local variable type inference" (da Java 10)

Costrutto `var`

- usabile nelle variabili locali (a funzioni/metodi) per avere maggiore concisione
- il compilatore capisce (inferisce) il tipo della varibile dall'espressione assegnata
- non abusarne, e comunque noi non lo useremo molto all'inizio del corso

```
1 var i = 5;                    // assegnamenti con tipi primitivi
2 var d = 145e-20;
3 var b = true;
4 var other = i;
5
6 var o = new Object();    // assegnamenti con classi
7 var s = "another string";
8 var p = new Point2D(10.4,20.3);
9 var q = p;                  // assegnamento da/a variabile
```

Oggetti e memoria

Gestione della memoria – entriamo nella JVM solo temporanemante

- tutti gli oggetti sono allocati nella memoria heap
- le variabili sono allocate nello stack, nei rispettivi record di attivazione
- le variabili di tipi primitivi contengono direttamente il valore
- le variabili che contengono oggetti in realtà hanno un riferimento verso lo heap
- nota: ancora non sappiamo cosa contiene un oggetto

Situazione relativa al codice della slide precedente

- le variabili `i`, `d`, `b`, `other` contengono valori
- tutte le altre contengono un riferimento ad un oggetto, nello heap
- solo `p` e `q` "(si) riferiscono (al)lo stesso oggetto"
- `on` contiene un riferimento speciale, `null`

Visibilità, ossia "scope", delle variabili

Definita una variabile, dove sarà visibile?

- Regole molto simili a quelle del C
- variabili dentro un blocco non sono visibili fuori
- differenza rispetto al C: variabili non inizializzate non sono utilizzabili!

Tempo di vita degli oggetti

- finita l'esecuzione delle istruzioni relative allo scope di una variabile, l'oggetto continua a esistere
- verrà deallocato automaticamente dal sistema se non più usato
  - se, direttamente o indirettamente, nessuna variabile lo può raggiungere
  - un componente della JVM, il garbage collector, è preposto a questo compito
  - ne vedremo il funzionamento a breve..

Outline

Costruire classi

Premesse

- la classe è l'unità fondamentale di programmazione OO
- progettare e costruire classi correttamente sarà l'obiettivo del corso
- incominciamo descrivendo la loro struttura generale
- daremo via via linee guida sempre più precise

Cos'è una classe

- è un template (stampino) per generare oggetti di una certa forma
- ne definisce tipo, struttura in memoria e comportamento

Classe vs. oggetto

- classe: è una descrizione (parte di programma)
- oggetto: è una entità a tempo di esecuzione, è istanza di una classe

Struttura di una classe

Nome della classe

è anche il nome del tipo

Membri della classe

Campi (in inglese, fields)

- descrivono la struttura/stato dell'oggetto in memoria

Metodi (in inglese, methods)

- descrivono i messaggi accettati e il comportamento corrispondente

(altri elementi che vedremo..)

Classi: un po' di codice

Costruzione classi

```
1 class A {    // A è il nome della classe
2   ...        // qui si riporta il suo contenuto
3 }
4
5 class AnotherClassExample { // Nota il CamelCase
6   ...
7 }
```

Uso

```
1 // codice cliente
2 A obj1 = new A();    // creo un oggetto di A, con nome obj1
3 A obj2 = new A();    // creo un altro oggetto di A
4 AnotherClassExample obj3 = new AnotherClassExample();
5 A obj4;              // variabile obj4 non inizializzata
6 obj4 = new A();      // ok
7 obj4 = new AnotherClassExample();// NO!! Errore semantico..
```

Campi

Elementi costitutivi dei campi

- i campi di una classe assomigliano ai membri di una struct del C
- ognuno è una sorta di variabile (nome + tipo)
  (per i campi non è usabile il costrutto `var`)
- ve ne possono essere 0,1, molti
- lo stato di un oggetto è l'attuale valore associato ai campi
- potrebbero essere valori primitivi, o altri oggetti

Valore di un campo

- impostabile al momento della sua dichiarazione
- se non inizializzato vale:
  - 0 per i tipi numerici
  - `false` per i booleani
  - `null` per le classi
- accessibile da codice cliente con notazione obj.field

Campi: un esempio "toy" (giocattolo)

Classe

```
1 class A {
2   int i;
3   int j = 2;
4   Object o;
5 }
```

Uso

```
1 A obj = new A();
2 int a = obj.i;    // a varrà 0
3 int b = obj.j;    // b varrà 2
4 obj.i = 10;       // modifico lo stato di obj
5 int d = obj.i;    // d varrà 10
6 obj.o = new Object();  // riassegno obj.o
7 A obj2 = new A();
8 obj.i = obj2.i;  // quanto varrà ora obj.i?
```

Campi: un esempio realistico, Point3D

Classe

```
1  class Point3D {
2    double x; // Nota, l'ordine dei campi è irrilevante
3    double y;
4    double z;
5  }
```

Uso

```
1  Point3D a = new Point3D();  // Creo due punti, di nome a e b
2  Point3D b = new Point3D();
3  a.x = 10.0; // Imposto lo stato di a
4  a.y = 20.0;
5  a.z = 30.0;
6  b.x = a.x * 2;  // Imposto lo stato di b
7  b.y = a.y * 2;  // .. a partire da quello di a
8  b.z = a.z * 2;
9  int mod2a = a.x * a.x + a.y * a.y + a.z * a.z;
10 int mod2b = b.x * b.x + b.y * b.y + b.z * b.z;
11 boolean aGreater = (mod2a > mod2b); // false
```

Metodi

Elementi costitutivi dei metodi

- i metodi di una classe assomigliano a funzioni (del C)
- ognuno ha una intestazione (signature) e un corpo (body)
  - a sua volta l'intestazione ha il nome, tipo di ritorno, argomenti
- di metodi ve ne possono essere 0,1, molti
- definiscono il comportamento dell'oggetto

Significato di un metodo

- codice cliente richiama un metodo con notazione obj.meth(args)
- ciò corrisponde a inviargli un messaggio
- `obj` è chiamato il receiver del messaggio (o della invocazione)
- il comportamento conseguente è dato dall'esecuzione del corpo
- il corpo può leggere/scrivere il valore dei campi

Metodi: esempio toy

Classe

```
1 class A {
2   int i;
3   void add(int a){ // input "int a"
4     i = i + a;
5   }
6   int getValue(){     // intestazione funzione
7     return i;            // corpo funzione
8   }
9 }
```

Uso

```
1 A obj = new A();
2 int v = obj.i;                    // vale 0
3 obj.add(10);                      // modifico obj
4 obj.add(20);                      // modifico obj
5 int v2 = obj.i;                   // vale 30
6 int v3 = obj.getValue();          // vale 30
```

La variabile speciale this

this

- dentro ad un metodo si può accedere agli argomenti o ai campi
- per rendere meno ambigua la sintassi, Java fornisce una variabile speciale denotata con this, che contiene il riferimento all'oggetto che sta gestendo il messaggio corrente
- per motivi di leggibilità, è opportuno usarla sempre in questo corso

```
1  class A {
2    int i;
3    void add(int a){
4      this.i = this.i + a;     // this.i: il "mio" campo i
5    }
6    int getValue(){
7      return this.i;
8    }
9    int get(){     // Un alias per getValue
10     return this.getValue();
11   }
12 }
```

Metodi: altro esempio Point3D

```
1  class Point3D {    // dichiarazione classe
2    double x;        // 3 campi
3    double y;
4    double z;
5    void build(double a,double b,double c){
6      this.x = a; this.y = b; this.z = c;
7    }
8    double getSquaredModulus(){
9      return this.x * this.x + this.y * this.y + this.z * this.z;
10   }
11   boolean isEqual(Point3D q){
12     return this.x == q.x && this.y == q.y && this.z == q.z;
13   }
14 }
15 ..
16 // codice cliente
17 Point3D p1 = new Point3D();        // creo un punto p1
18 p1.build(10.0,20.0,30.0);         // ne imposto i valori
19 Point3D p2 = new Point3D();        // creo un punto p2
20 p2.build(10.0,20.0,31.0);         // ne imposto i valori
21 double m2 = p1.getSquaredModulus();// ottengo il mod.quad. di p1
22 boolean b = p1.isEqual(p2);          // chiedo a p1 se è uguale a p2
```

Outline

Programmi Java

Elementi costitutivi dei programmi Java

- librerie di classi del Java Development Kit
- librerie esterne (nostre o di altri)
- un insieme di classi che costituiscono l'applicazione
- almeno una di tali classi ha un metodo speciale `main`
- un `main` è il punto d'accesso di un programma

Il main

Il `main` deve avere la seguente dichiarazione:

- `public static void main(String[] args){..}`

tre concetti che spiegheremo in dettaglio nel prosieguo:

- `public` indica il fatto che debba essere visibile "a tutti"
- `static` indica che non è un metodo dell'oggetto, ma della classe
- `String[]` indica il tipo "array di stringhe"

Package e librerie Java: organizzazione

Librerie di Java

- Documentazione auto-generata, consultabile offline, o online:
  `https://docs.oracle.com/en/java/javase/17/docs/api/`
  (google search "`javadoc 17`")
- contano 4000+ classi, raggruppate in 200+ package (e 50+ moduli)

Package

- un package è un contenitore di classi con uno scopo comune, di alto livello
- tipicamente un package contiene qualche decina di classi
- i package sono organizzati ad albero, con notazione `nome1.nome2.nome3`
- Package principali: `java.lang`, `java.util`, `java.io`

Package, moduli e librerie Java: moduli

Moduli (Java 9+)

- un modulo definisce un frammento di codice "autonomo":
  - testabile, distribuibile, con chiara interfaccia e dipendenze da altri
- p.e., `java.base`, `java.desktop`, `java.sql`, `java.xml`
- librerie esterne compatibili con Java 9+ sono distribuite in uno o più moduli
- un modulo è costituito internamente da uno o più package
- p.e., `java.base` contiene i package principali che useremo

Impatto sulla programmazione "base"

- il concetto di modulo non impatta i sorgenti, ma solo il "project management": per il momento non ce ne occuperemo perché il JDK fornisce "di default" accesso a tutti i moduli che ci servono (`java.*`)
- il concetto di package invece impatta i sorgenti: il nome completo di una classe dipende dal package in cui si trova

Package, moduli e librerie Java: uso

Importare una classe di "libreria"

- per usare le classi di una libreria prima le si importa
- lo si fa con la clausola `import`, da usare all'inizio del sorgente
  - importo la singola classe: `import java.util.Date;`
  - importo l'intero package: `import java.util.*;`
  - importazione di default: `import java.lang.*;`
- senza importazioni, ogni classe andrebbe sempre qualificata indicandone anche il package completo:

  ```
  java.util.Date obj = new java.util.Date();
  ```
- importare evita quindi solo di dover indicare ogni volta il package

Classi (e funzionalità) "deprecate"

- dichiarate come "scadute", ossia preferibilmente "da non usare più" (legacy)
- noi le utilizzeremo a volte per scopi didattici, solo a inizio corso
- p.e. `java.util.Date`

Stampe su schermo

La procedura di stampa `System.out.println`

- `System` è una classe nel package `java.lang`
- `out` è un suo campo statico, rappresenta lo standard output
- `println` è un metodo che accetta una stringa e la stampa
- l'operatore + concatena stringhe a valori

```
1  class A {
2    int i;
3    void print(){
4      System.out.println("I'm an object of class A");
5      System.out.println("My field value is: " + this.i);
6    }
7  }
8  ..
9  A obj = new A();
10 obj.i = 12;
11 obj.print();
```

"Hello world" Java Program

Hello.java

```java
// This class necessarily goes into a file Hello.java
class Hello{
  public static void main(String[] args){
    System.out.println("Hello World!");
  }
}
```

Compilazione ed esecuzione

- con un editor di testo si scrive la classe in un file `Hello.java`
- si compila la classe col comando: `javac Hello.java`
- se non ci sono errori, viene generato il bytecode `Hello.class`
- si esegua il programma con: `java Hello`
- la JVM cerca la classe `Hello`, e ne esegue il `main`

Librerie, oggetti, e stampe

```java
import java.util.Date;
import java.util.Random;
// java.lang.System needs no import

class PrintingObjects{
  public static void main(String[] args){
    Date d = new Date();
    System.out.println("Current date: " + d);
    System.out.println("Millisec. from 1/1/1970: " + d.getTime());

    Random r = new Random();
    System.out.println("Random number: " + r.nextInt());
    System.out.println("Another random number: " + r.nextInt());
    System.out.println("N. random in (0-99): " + r.nextInt(100));
    String vjava = System.getProperty("java.version");
    String osname = System.getProperty("os.name");
    String usrdir = System.getProperty("user.dir");
    System.out.println("Java version: " + vjava);
    System.out.println("OS Name: " + osname);
    System.out.println("Usr dir: " + usrdir);
  }
}
```

Costruire e provare classi

```
1  class Point3D {
2    double x;
3    double y;
4    double z;
5    void build(double a,double b,double c){
6      this.x = a; this.y = b; this.z = c;
7    }
8    double getSquaredModulus(){
9      return this.x * this.x + this.y * this.y + this.z * this.z;
10   }
11   boolean isEqual(Point3D q){
12     return this.x == q.x && this.y == q.y && this.z == q.z;
13   }
14 }
```

Classi, e classi clienti

```java
class Point3D {
  double x;
  double y;
  double z;
  void build(double a,double b,double c){
    this.x = a; this.y = b; this.z = c;
  }
  double getSquaredModulus(){
    return this.x * this.x + this.y * this.y + this.z * this.z;
  }
  boolean isEqual(Point3D q){
    return this.x == q.x && this.y == q.y && this.z == q.z;
  }
}
```

```java
class UsePoint3D{
  public static void main(String[] args){
    Point3D p1 = new Point3D();  // creo un punto p1
    p1.build(10.0,20.0,30.0);    // ne imposto i valori
    Point3D p2 = new Point3D();  // creo un punto p2
    p2.build(10.0,20.0,31.0);    // ne imposto i valori
    System.out.println("Squared modulus of p1: " + p1.getSquaredModulus());
    System.out.println("is p1 equal to p2? : " + p1.isEqual(p2));
  }
}
```

- si compilano separatamente, o con: `javac *.java`
- si esegue a partire dalla classe col `main`: `java UsePoint3D`

Preview del prossimo laboratorio

Obiettivi

- familiarizzare con la compilazione da linea di comando in Java
- fare qualche esercizio con la costruzione e uso di classi