UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Laboratorio di Programmazione ad Oggetti

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

Sommario (1)

- Tecnologia Java: Linguaggio e Piattaforma
- Java Virtual Machine
- Storia
- Struttura JDK, tools, documentazione
- Principi base del paradigma O.O.
  - Astrazione
  - Incapsulamento e Information Hiding
  - Modularità
  - Gerarchia
  - Polimorfismo
- Oggetto e Classe

Sommario (2)

- Struttura di un programma Java
- First Cup of Java
- Commenti e Spazi Bianchi
- Identificatore e Letterale
- Blocco e statement
- Tipi
  - Primitivi
  - Reference
- Variabili e scope
- Rappresentazione dati
- Passaggio per valore

Risorse per J2SE (1)

- Home Page
  - https://www.oracle.com/java/technologies/
- Documentazione
  - https://docs.oracle.com/en/java/javase/index.html
- Java Tutorial
  - https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html
  - Download: https://www.oracle.com/java/technologies/javase/java-tutorial-downloads.html
- Specifica del linguaggio
  - https://docs.oracle.com/javase/specs/

Risorse per J2SE (2)

- Titolo: Manuale di Java 9
  Autore: Claudio De Sio Cesari
  Casa Editrice: Hoepli
  ISBN-10: 8820383020

Risorse per J2SE (3)

- Titolo: Thinking in Java 4 ed. Volume 1- I fondamenti
  Autore: Bruce Eckel
  Casa Editrice: Pearson
  ISBN: 8871923030
- Titolo: Thinking in Java 4 ed. Volume 2 – Tecniche avanzate
  Autore: Bruce Eckel
  Casa Editrice: Pearson
  ISBN: 8871923049
- Titolo: Thinking in Java 4 ed. Volume 3 - Concorrenza e interfacce grafiche
  Autore: Bruce Eckel
  Casa Editrice: Pearson
  ISBN: 8871923057

Ambienti di Sviluppo (IDE)

- Free
  - **Eclipse**
    http://www.eclipse.org/
  - NetBeans
    http://netbeans.apache.org/
    http://www.netbeans.org/
  - Visual Studio Code
    https://code.visualstudio.com/
- Commerciali
  - Intellij IDEA
    http://www.jetbrains.com/idea/

Meccanismo di rilascio e licenza (1)

- A **marzo** e **settembre** di ogni anno vengono rilasciate delle **feature release** di Java: nuove funzionalità
- Numerazione delle feature release è progressiva
- Ogni release principale riceverà solo due aggiornamenti (**update release**)
- Esempio
  - Versione 11 rilasciata a settembre 2018
  - Aggiornamenti in ottobre 2018 e gennaio 2019 → ultima versione è la 11.0.2
- Update release per risolvere piccoli problemi o aggiungere miglioramenti secondari
- Versione attuale 15.0.2

Meccanismo di rilascio e licenza (2)

Version	class file format version[8]	Release date	End of Free Public Updates[9][10][11][12][13][14][15]	Extended Support Until
Java SE 9	53	21th September 2017	March 2018 for OpenJDK	—
Java SE 10	54	20th March 2018	September 2018 for OpenJDK	—
Java SE 11 (LTS)	55	25th September 2018	OpenJDK currently maintained by Red Hat[20] September 2026 for Azul[12] October 2024 for IBM Semeru[14] At least October 2024 for Eclipse Adoptium[10] At least September 2027 for Amazon Corretto[11] At least October 2024 for Microsoft[21][15]	September 2026 for Oracle[9] September 2026 for Azul[12] October 2024 for Red Hat[13]
Java SE 12	56	19th March 2019	September 2019 for OpenJDK	—
Java SE 13	57	17th September 2019	OpenJDK currently maintained by Azul[22] March 2023 for Azul[12]	—
Java SE 14	58	17th March 2020	September 2020 for OpenJDK	—
Java SE 15	59	16th September 2020	OpenJDK currently maintained by Azul[23] March 2023 for Azul[12]	—
Java SE 16	60	16th March 2021	September 2021 for OpenJDK	—
Java SE 17 (LTS)	61	14th September 2021	OpenJDK currently maintained by SAP[24] September 2029 for Azul[12] October 2027 for IBM Semeru[14] At least September 2027 for Microsoft[15] At least September 2027 for Eclipse Adoptium [10]	September 2029 or later for Oracle[9] September 2029 for Azul[12] October 2027 for Red Hat[13]
Java SE 18	62	22th March 2022	September 2022 for OpenJDK and Adoptium	—
Java SE 19	63	20th September 2022	March 2023 for OpenJDK	—
Java SE 20	—	March 2023	September 2023 for OpenJDK	—
Java SE 21 (LTS)	—	September 2023	September 2028	September 2031 for Oracle[9]
Legend: Old version	Older version, still maintained	**Latest version**	Future release	

https://en.wikipedia.org/wiki/Java_version_history

Meccanismo di rilascio e licenza (3)

- A partire da aprile 2019 è stato modificato il contratto di licenza di utilizzo di Oracle JDK
- Feature release saranno rilasciate con licenza “GPL + Classpath Exception” (OpenJDK)
  - https://jdk.java.net/
- Long Term Release (LTS) saranno rilasciate tramite Oracle JDK quindi con una licenza che non è libera né gratuita
  - Si può utilizzare per l’uso e lo sviluppo personali senza alcun costo
  - https://www.oracle.com/technetwork/java/javase/downloads/index.html
- OpenJDK
  - Comunità che permette di collaborare ad una implementazione open-source di Java SE
  - Contiene **codice sorgente** di Java SE e i progetti che ne guidano l’evoluzione
  - Sito: **https://openjdk.java.net/**

Alternative

- Adopt JDK:
  - https://adoptopenjdk.net/
- Amazon Corretto
  - https://aws.amazon.com/it/corretto/
- Azul Zulu
  - https://www.azul.com/downloads/zulu/

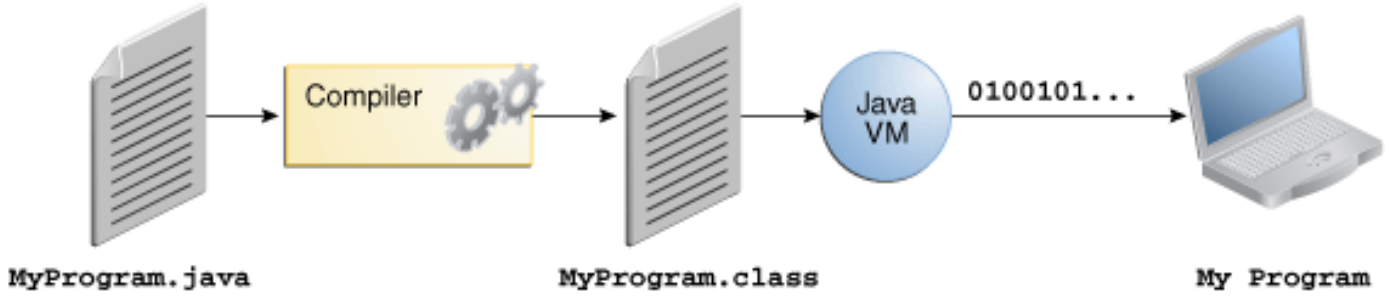
Tecnologia Java (1)

- Linguaggio
  - Sun Microsystems definisce Java come
    un linguaggio ***semplice e familiare, orientato agli oggetti, distribuito, robusto, sicuro, architettura neutrale, portabile,*** *ad* ***alte prestazioni, interpretato, multithreaded*** *e* ***dinamico***

    [J. Gosling e H. McGilton, “The Java Language Environment. A White Paper.” Ottobre 1995]

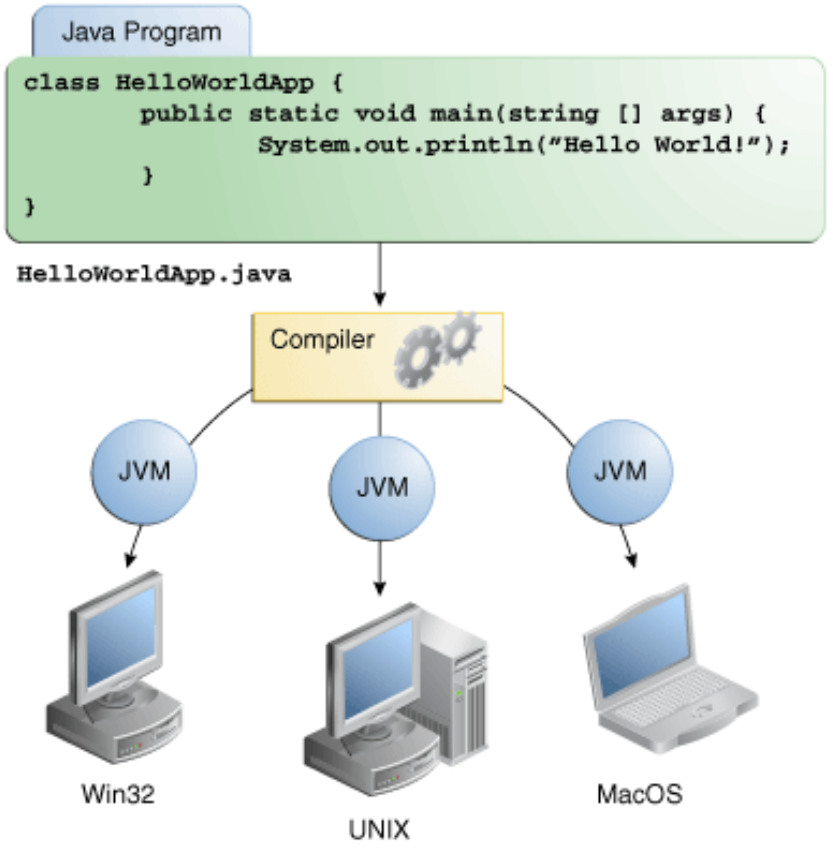
Tecnologia Java (2)

- Sorgente viene compilato in un linguaggio intermedio indipendente dalla piattaforma detto bytecode
- Interprete traduce il bytecode in istruzioni macchina del computer



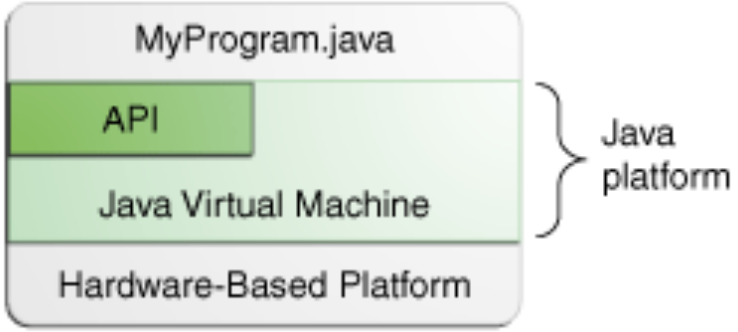
Tecnologia Java (3)

- Bytecode sono istruzioni macchina per la Java Virtual Machine (JVM)
- Bytecode rende possibile **write once run anywhere**
- E' possibile eseguire lo stesso codice su Windows, Linux, MacOS, qualsiasi dispositivo che ha una Java VM



Tecnologia Java (4)

- Piattaforma
  - Ambiente hardware e software all'interno del quale vengono eseguiti i programmi
  - E' composta dalla
    - Java Virtual Machine (JVM)
    - Java Application Programming Interface (Java API)



Java Virtual Machine (1)

- Costituisce la pietra angolare della piattaforma Java
- E' il componente responsabile dell'indipendenza dall'Hardware e dal sistema operativo
- Implementazioni correnti della VM emulano la VM su Windows, Linux e Mac OS
- Può essere implementata direttamente su HW

Java Virtual Machine (2)

- E' fondamentalmente un computer astratto che ha un
  - Formato di file delle classi
  - Tipi di dato: primitivi e reference
  - Insieme di istruzioni per una ipotetica CPU
  - Aree dati durante esecuzione
    - Registro PC (Program counter): per thread
    - Stack per thread: Elemento frame
    - Heap: globale
    - Method Area
      - Memorizza strutture dati per classi come run-time constant pool, dati e metodi static, codice per i metodi e costruttori methods and constructors

JVM: startup e esecuzione (1)

- JVM inizia esecuzione invocando il metodo `main` della classe specificata e passando come singolo array di stringhe gli argomenti specificati
  - `java Test reboot Bob Dot Enzo`
- Load classe `Test`
  - Se classe non è caricata in memoria (ovvero JVM non ha una rappresentazione binaria) JVM usa il **class loader** per caricarla
  - VM usa tre class loader
    - **Bootstrap**: compito di caricare classi presenti in `rt.jar` e altre librerie core che si trovano in `$JAVA_HOME/jre/lib`
    - **Extension**: compito di caricare classi presenti in `$JAVA_HOME/jre/lib/ext`
    - **Application**: compito di caricare classi dell'applicazione ovvero quelle specificate nel classpath, nella variabile d'ambiente `CLASSPATH`
    - A partire da Java 9 con l'introduzione dei moduli il meccanismo è cambiato

JVM: startup e esecuzione (2)

....

- **Garbage Collector**: compito di rimuovere oggetti (create con `new`) non più utilizzati. Si può invocare tramite `System.gc()` ma l'esecuzione non è garantita
- **Java Native Interface (JNI)**: interagisce con le librerie dei metodi nativi e fornisce le librerie native richieste per l'esecuzione
- **Native Method Libraries**: Collezione di librerie native utilizzate dall'execution engine

Storia (1)

- Nascita ’91 Patrick Naughton e James Gosling progettarono linguaggio per dispositivi consumer (switchbox per televisione via cavo)
  - Progetto Green
  - Linguaggio Oak
    - JVM da idea di Niklaus Wirth
    - Basato su C++
- ’92 progetto Green rilasciò prodotto “*7”
  - Non riscosse successo
- ’94 progetto First Person (i.e. Green) chiuse i battenti
- Metà ’94 Gosling&C. si resero conto che con l’avvento del web dovevano realizzare un browser portabile e sicuro (HotJava con applet)
- 23 maggio 1995 SUNWorld ’95 presentarono la nuova tecnologia

Storia (2)

- Inizi ’96 SUN rilasciò la prima versione di Java 1.0.2
- Dopo poco 1.1 che introduceva notevoli migliorie
- 1998 alla conferenza JavaOne rilasciarono versione 1.2 ovvero Java 2 Standard Edition Software Development Kit Version 1.2
- ...
- 1.3
- 1.4
- JavaOne 2004 presentata 1.5 ovvero Java 5.0
- ...
- Java SE 15, Settembre 2020

Evoluzione del linguaggio

Java Development Kit	Codename	Release	Classes
Java SE 15		09/2020	
…	…	….	…
Java SE 11	---	09/2018	4411
Java SE 10	---	03/2018	6002
Java SE 9	---	09/2017	6,005
Java SE 8	---	2014	4,240
Java SE 7	Dolphin	2011	4,024
Java SE 6	Mustang	2006	3,793
Java 2 SE 5.0 with JDK 1.5.0	Tiger	2004	3,279
Java 2 SE with SDK 1.4.0	Merlin	2002	2,991
Java 2 SE with SDK 1.3	Kestrel	2000	1,842
Java 2 SE with SDK 1.2	Playground	1998	1,520
Development Kit 1.1	----	1997	504
Development Kit 1.0	Oak	1996	212

Tool: javac (1)

- Legge la definizione di classi ed interfacce scritte nel linguaggio Java e le compila in bytecode
- Sintassi
  - `javac [ options ] [ sourcefiles ]`
- Vi sono due modi per passare i nomi dei file sorgenti
  - Per un numero piccolo di file si inserisce la lista dei file nella riga di comando
  - Per un numero maggiore si può creare un file e si inserisce tale lista separando gli elementi con spazi o invio a capo (`javac @nomefile`)
- Nomi dei file devono avere suffisso `.java`
- Classi compilate suffisso `.class`

https://docs.oracle.com/en/java/javase/11/tools/tools-and-command-reference.html

Tool: javac (2)

- Ricerca classi
  - Se durante la compilazione (o esecuzione) di un file si fa *riferimento* ad una classe/interfaccia che non si trova all'interno del file il compilatore effettua la seguente ricerca
    - Bootstrap
    - Estensione
    - Class path definito dall'utente
      - Variabile d'ambiente `CLASSPATH`
      - Opzione `-classpath`

Tool: java

- Esegue un programma Java
- Sintassi

```
java [options] classname [args]
java [options] -jar filename [args]
```

- `options`: opzioni a riga di commando separate da spazio
- `classname`: nome della classe contenente il main
- `filename`: nome del file Java Archive (JAR) file (solo con opzione `-jar`)
- `args`: argomenti passati al metodo `main()` separati da spazio

Java API Documentation

Download: https://www.oracle.com/java/technologies/javase-jdk11-doc-downloads.html

https://docs.oracle.com/en/java/javase/11/docs/api/index.html

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java SE 11 & JDK 11

ALL CLASSES — SEARCH: Search

Java® Platform, Standard Edition & Java Development Kit
Version 11 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules | **Java SE** | **JDK** | **Other Modules**

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.
java.prefs	Defines the Preferences API.
java.rmi	Defines the Remote Method Invocation (RMI) API.
java.scripting	Defines the Scripting API.
java.se	Defines the API of the Java SE Platform.
java.security.jgss	Defines the Java binding of the IETF Generic Security Services API (GSS-API).

Principi base del paradigma O.O.

- Astrazione
- Incapsulamento & Information Hiding
- Modularità
- Gerarchia
- Polimorfismo

Astrazione

"Qualsiasi modello che include gli aspetti più importanti o essenziali di qualcosa mentre ***ignora i dettagli meno importanti****, immateriali. Il risultato è di* ***rimuovere le differenze*** *ed* ***enfatizzare gli aspetti comuni"***

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Astrazione

- Permette di gestire la complessità concentrandosi sulle caratteristiche essenziali di un'entità che la distingue dalle altre
- E' dipendente dal dominio e dalla prospettiva, cioè quello che è importante in un contesto potrebbe non esserlo in un altro
  - def. macchina per un venditore diversa da quella di un progettista
- OO modella il sistema utilizzando l'astrazione (esempio classi)

Incapsulamento & Information Hiding

“La localizzazione fisica di caratteristiche (es. proprietà, comportamento) in una singola scatola nera che nasconde l’implementazione (e le relative decisioni di design) dietro una interfaccia pubblica”

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Incapsulamento & Information Hiding

- Incapsulamento
  - Collegare i dati insieme con le operazioni che agiscono su di essi
- Information Hiding
  - Nascondere ai possibili client (utilizzatori) le scelte interne di design e gli effetti che eventuali cambiamenti di tali decisioni comportano
- Incapsulamento è una **facility** del linguaggio mentre l'Information Hiding è un **principio di design**

Modularità

"La decomposizione fisica e logica di cose (es. responsabilità e software) in gruppi piccoli e semplici (es. requisiti e classi rispettivamente) che incrementa il raggiungimento degli obiettivi dell'ingegneria del software"

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Modularità

- Per gestire la complessità si può suddividere qualcosa che è complesso in pezzi più piccoli che sono più maneggevoli
- O.O. modella la modularità con package e subsystem

Gerarchia

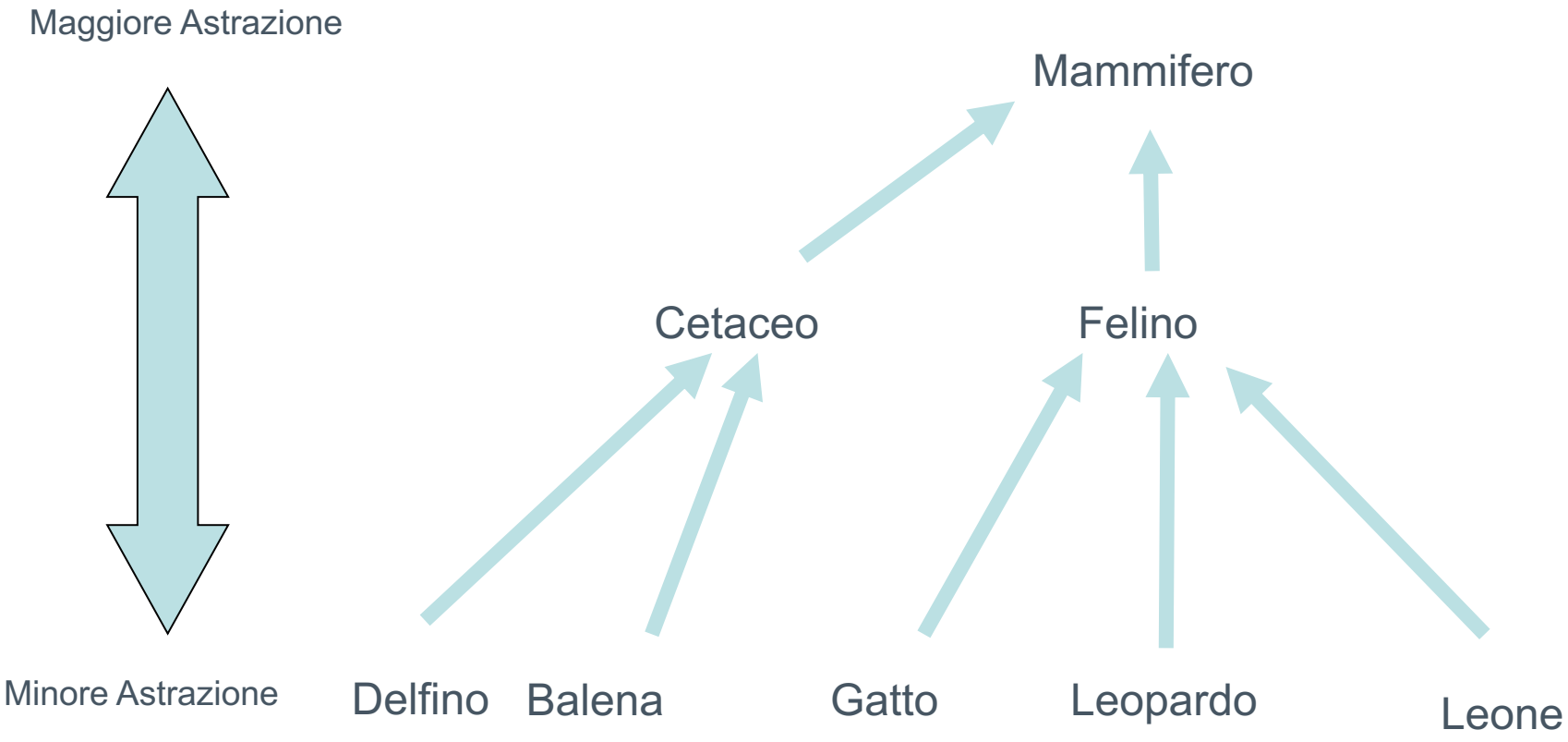
“Qualsiasi graduatoria (ranking) o ordine di astrazione in una struttura ad albero”

[Dizionario di Object Technology – Firesmith, Eykholt 1995]

Gerarchia

- Permette di organizzare un qualcosa in base ad un certo ordine
  - Esempi: complessità, responsabilità
- Descrive differenze o variazioni di un particolare concetto

Gerarchia



***Fratelli* hanno stesso livello di astrazione**

Polimorfismo

- Linguaggi procedurali (esempio Pascal) basati su idea che procedure e funzioni, e i loro operandi, hanno un unico tipo
- Tali linguaggi sono detti *monomorphic*, cioè ogni valore e variabile può avere uno ed un solo tipo
- Linguaggi O.O. sono detti *polymorphic*, cioè i valori e le varibili possono avere più di un tipo
- Dal greco *polymorphos* “avere molte forme”

Punti di forza dell'O.O.

- Singolo paradigma
- Facilita il riuso di codice e di architetture
- Modelli riflettono maggiormente la realtà
  - Descrizione più accurata dei dati e processi
  - Decomposizione basata su partizionamento naturale
  - Più facile da comprendere e mantenere
- Stabilità
  - Piccolo cambiamento nei requisiti non significa massicci cambiamenti nel sistema durante lo sviluppo

Oggetto (1)

- Informalmente rappresenta
  - entità fisica: trattore
  - entità concettuale: processo chimico
  - oppure entità software: lista, coda...
- Formalmente
  - Manifestazione concreta di un'astrazione
  - Entità con un confine e un'*identità* ben definite che incapsula *stato* e *comportamento*
  - Istanza di una classe
- Es.: Ferrari di Vettel e Leclerc, Mio computer

Oggetto (2)

- Stato
  - Possibile condizione nel quale l'oggetto potrebbe esistere e generalmente cambia nel tempo
  - Implementato mediante proprietà (attributi) con valori, e collegamenti ad altri oggetti
- Comportamento
  - Determina come un oggetto agisce e reagisce alle richieste di un altro oggetto
  - Rappresentato dall'insieme di messaggi a cui può rispondere (operazioni)
- Identità
  - Rende possibile la distinzione tra due oggetti anche se hanno lo stesso stato e lo stesso valore nei suoi attributi

Classe

- Descrizione di un gruppo di oggetti con proprietà (attributi), comportamento (operazioni), relazioni e semantica comuni
  - Un oggetto è una istanza di una classe
- Astrazione che
  - Enfatizza caratteristiche rilevanti
  - Sopprime le altre caratteristiche

- *Principio O.O. Astrazione*

Esempio di classe

- Nome
  - Corso
- Proprietà
  - Nome, Luogo, Durata, Crediti, Inizio, Fine
- Comportamento
  - Aggiunta studente
  - Cancellazione studente
  - Verifica se è pieno

Relazione tra classe ed oggetto

- Classe è una definizione astratta di un oggetto
  - Definisce la struttura e il comportamento di ogni oggetto nella classe
  - Serve come *template* per creare oggetti
- Oggetti sono *raggruppati* in classi

Struttura programma Java

- Ogni programma Java e composto da una o più classi
- Una classe contiene uno o più metodi
- Un metodo contiene le istruzioni
- Punto di partenza di un'applicazione Java classe contenente un metodo `main`
- Sintassi

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

}
```

First Cup of Java (1)

- Requisito: avere installato JDK
- Creare un file sorgente contenente l'applicazione che stampa "Hello World"
- Compilare il file per generare l'*eseguibile* (bytecode)
- Eseguire il bytecode con l'interprete java

First Cup of Java (2)

Commento

```
/**
  * The HelloWorldApp class implements an application *
that displays "Hello World!" to the standard
  * output.
*/
```

Classe

Main

```
public class HelloWorldApp {
   public static void main(String[] args) {
         // Display "Hello World!“
         System.out.println("Hello World!");
   }
}
```

Struttura di una classe (1)

```
<modificatore> class <nome della classe> [extends Tipo]
                    [implements ListaTipi] {


[<dichiarazione di attributi>]

[<dichiarazione dei costruttori>]

[<dichiarazione dei metodi>]


}
```

Corpo della classe

Struttura di una classe (2)

› Metodo

```
<modificatore> <tipo ritorno> <identificatore> ([lista parametri]) {
      [<istruzioni>]
}
```

› Esempio

```
public class Test {
   public static int somma(int x, int y) {
        return x + y;
   }
   public static void main(String[] args) {
      int s = somma(4, 5);
      System.out.println(s);
   }
}
```

Struttura di una classe (3)

› Esempio 2

```
public class Punto {
    public int x;
    public int y;
}
```

Struttura di una classe (4)

```
….
public class TestOggettiPunto {
    public static void main(String args[]) {
        Punto punto1;
        punto1 = new Punto();
        punto1.x = 2;
        punto1.y = 6;
        Punto punto2 = new Punto();
        punto2.x = 0;
        punto2.y = 1;
        System.out.println(punto1.x);
        System.out.println(punto1.y);
        System.out.println(punto2.x);
        System.out.println(punto2.y);
    }
}
```

Struttura di una classe (5)

› Esempio 3

```
public class Auto {
    public int numeroRuote = 4;
    public int cilindrata; // quanto vale?
    public void muoviti() {
        // implementazione del metodo...
    }
}
```

Struttura di una classe (6)

```
…..
public class TestOggettiAuto {
    public static void main(String args[]) {
        Auto fiat600;
        fiat600 = new Auto();
        fiat600.cilindrata = 1100;
        fiat600.muoviti();
        Auto california = new Auto();
        california.cilindrata = 4300;
        california.muoviti();
    }
}
```

Invocazione di metodi (1)

› Esempio

Parametri formali

```
public class Aritmetica {
    public int somma(int a, int b) {
        return (a + b);
    }
}
public class TestSomma {
    public static void main(String args[]) {
        Aritmetica oggettoAritmetica = new Aritmetica();
        int risultato = oggettoAritmetica.somma(5, 6);
        System.out.println(risultato);
    }
}
```

Parametri attuali

Invocazione di metodi (2)

› Esempio 2

```
public class Saluti {
    public void stampaSaluto() {
        System.out.println("Ciao");
    }
}
public class TestSaluti {
  public static void main(String args[]) {
    Saluti oggetto1 = new Saluti();
    oggetto1.stampaSaluto();
    int i = oggetto1.stampaSaluto(); //Cosa succede qui?
  }
}
```

Commenti (1)

- Sono spesso detti documentazione in-line
- Sono inclusi per documentare lo scopo e le funzionalità del programma
- Non ne influenzano il funzionamento
- Vengono trascurati dal compilatore
- Possono avere tre forme

```
// commenti fino alla fine della riga
/* commento che
   può stare su piu righe */
/** commento che genera documentazione */
```

Commenti (2)

- Commenti non si possono annidare
- Si possono utilizzare `/*` e `*/` all'interno di `//`
- Si può utilizzare `//` all'interno dei commenti `/*` e `/**`
- All'interno di `/*` e `/**` e possibile utilizzare `/*` ma non `*/` contemporaneamente
- Esempio
  - `/* this comment /* // /** ends here: */`

Spazi bianchi

- Spazi, righe vuote e le tabulazioni sono considerati spazi bianchi
- Sono usati per separare le parole e i simboli di un programma
- Vengono ignorati dal compilatore
- Vengono utilizzati per formattare un programma Java migliorando la leggibilità

Blocchi

- Collezione di statements oppure dichiarazione locale di classi oppure dichiarazione di variabili locali racchiusi tra parentesi graffe
- Utilizzati nella definizione di classi, metodi, `if`, `while`, `for`, ...
- E' possibile annidare blocchi
- Esempio

```
{
    a = b + c;
    d += 10;
}
```

Identificatore

- Sequenza di lunghezza non limitata di *lettere* e *cifre* il cui primo carattere della sequenza deve essere una **lettera** oppure `_` o **`$`**
- Lettera è un carattere per cui il metodo `Character.isJavaIdentifierStart` ritorna `true`
- Cifra è un carattere dove il metodo `Character.isJavaIdentifierPart` ritorna `true`
- Vengono utilizzati per i nomi delle variabili, delle classi, dei metodi e sono **case-sensitive**
- Non identificatori le keyword e i letterali `true`, `false` e `null`
- Esempi
  - `Prodotto`
  - `prodotto`
  - `$userName`

Keyword riservate Java 8

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

const e goto riservate

Keyword riservate Java 11

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

_ *(underscore)*

Statements

- Statement una o più linee di codice terminante con un `;`
- Esempi

```
result = x * y;
x = new Point();
int x = 1 + 3 *
6 + (12 + 5);
```

Letterale

- Porzione di codice che rappresenta il valore di un tipo primitivo, Stringa oppure `null`
- Tipi
  - Intero
  - Floating
  - Booleano: `true` o `false`
  - Carattere
  - String
  - `null`

Letterale: Intero (1)

- Tipo è `int`
- Se utilizzato come suffisso `L` oppure `l` diventa `long`
- Espresso
  - decimali (base 10): numero
  - ottale (base 8): `0`numero
  - esadecimale (base 16): `0x`numero
  - binario (base 2): `0b`numero
- Esempi
  - Decimale int: `10`
  - Decimale long: `10L`
  - Ottale int: `010`
  - Esadecimale int: `0x10`
  - Binario int: `0b10`

Letterale: Intero (2)

- Più grande letterale decimale di tipo `int` è `2147483648` `(2.147.483.648)` (2^{31})
- Da `0` to `2.147.483.647` possono essere utilizzati ovunque un letterale `int` può apparire
- `2147483648` può apparire soltanto come numero negativo
- Più grande decimale letterale di tipo `long` è `9223372036854775808L` (2^{63}=`9.223.372.036.854.775.808`)

Letterale: Intero (3)

- Più grande letterale positivo di tipo `int` esadecimale ed ottale sono `0x7fffffff` e `017777777777` rispettivamente (`2147483647`)
- Più grande letterale negativo di tipo `int` esadecimale ed ottale sono `0x80000000` e `020000000000` che rappresentano –`2147483648`
- `0xffffffff` e `037777777777` rappresentano –`1` in esadecimale ed ottale rispettivamente

Letterale: Floating (1)

- Composto da
  - Parte numero intero
  - Punto decimale (`.`)
  - Parte frazionaria
  - Esponente: `E` oppure `e` seguito da un intero con segno
  - Suffisso
    - F oppure f float
    - `D` oppure d double (default)
- Esempi
  - Double: `1e1, 2., .3, 0.0, 3.14`
  - Float: `1e1f, 2.f, .3f, 0f, 3.14f`

Letterale: Floating (2)

- Letterale positivo float più grande è `3.40282347e+38f`
- Letterale positivo float finito non zero più piccolo è `1.40239846e-45f`
- Letterale positivo double più grande è `1.79769313486231570e+308`
- Letterale positivo double finito non zero più piccolo è `4.94065645841246544e-324`

Letterale con Underscore

- E' possibile utilizzare _ all'interno di un numero
- Non si può usare
  - Inizio o alla fine di un numero
  - Adiacenti ad un punto decimale per i numeri in virgola
  - Prima dei suffissi
  - Nelle posizioni dove ci si aspetta una stringa di caratteri
- Esempi

```
float piGrego = 3.14_15F;
long bytesEsadecimali = 0xFF_EC_DE_5;
int x1 = 5_2;
int x2 = _52; //Errore violata regola 1
int x3 = 52_; //Errore violata regola 1
Float pi1 = 3_.1415F; //Errore violata regola 2
```

Letterale: Carattere

- Espresso come un carattere o una sequenza di *escape* racchiusa tra apici singoli
- Tipo è sempre `char`
- Esempi
  - `'c'`
  - `'\\'`
  - `'\n', '\t', '\b', '\r', '\'', '\"'`
  - `'\u000a' // la fine di una riga`

Letterale: String

- Espresso come una sequenza di zero o più caratteri racchiusi tra doppi apici
- Ogni carattere può essere rappresentato con una sequenza di escape
- Esempi
  - `"welcome"`
  - `"\""`
  - `"\n"`
  - `"ciao" + "ciao"`

Tipi di dato

- Consente di esprimere la natura del dato
- Indica il modo con cui verrà interpretata la sequenza di bit che rappresenta il dato
- La stessa sequenza può rappresentare un intero o un carattere
- Determina il campo dei valori che un dato può assumere
- Specifica le operazioni possibili sui dati

Tipi di dato

- Java è un linguaggio **fortemente tipato**
- Il tipo di ogni **variabile** o espressione può essere identificato leggendo il programma ed è già noto al momento della **compilazione**
- E' obbligatorio dichiarare il tipo di una variabile prima di utilizzarla
- Dopo la dichiarazione non è possibile assegnare alla variabile valori di tipo diverso

Tipi

- Ogni tipo di dato ha
  - Un *nome*
    - `int, double, char`
  - Un *insieme di valori letterali* possibili
    - `3, 3.1, 'c'`
  - Un *insieme di operazioni* lecite
    - `+, *, /, %, ……`
- Java ha
  - Tipi Primitivi
  - Tipi Reference

Tipi primitivi

- Logici
  - `boolean`
- Numerici
  - Integrale
    - `byte`, `short`, `int`, `long` e `char`
  - Floating
    - `double` e `float`

Tipi primitivi: boolean

- Valore `boolean` rappresenta condizione di verità o falsità
- Variabile di tipo `boolean` può rappresentare un valore a due stati
  - come un interruttore che è acceso o spento
- Letterali `true` e `false` sono unici valori ammessi
- Esempio
  - `boolean b = false;`

Tipi primitivi: char

- Rappresentato mediante schema di codifica Unicode
  - Java 1 utilizza Unicode 10.0
- E' composto da 2 byte senza segno (65535)
- Primi 128 caratteri della codifica Unicode sono caratteri ASCII
- Eccetto che per commenti, identificatori, letterali di caratteri e stringa tutti gli altri elementi sono caratteri ASCII

Tipi primitivi: char

Caratteri	Nome	Unicode
`\b`	Backspace	`\u0008`
`\t`	Tabulazione	`\u0009`
`\n`	Avanzamento riga	`\u000a`
`\r`	Invio a capo	`\u000d`
`\"`	Doppi Apici	`\u0022`
`\'`	Apici Singoli	`\u0027`
`\\`	Backslash	`\u005c`

Tipi primitivi: byte, short, int e long

› Rappresentazione avviene mediante notazione in complemento a due

Tipo	Spazio Memoria	Range
byte	1 byte	-128 a 127
short	2 byte	-32678 a 32687
int	4 byte	-2147483648 a 2147483647
long	8 byte	-9223372036854775808 a 9223372036854775807

Tipi primitivi: float e double

- Definiscono numeri con parti frazionarie
- double vengono detti numeri a doppia precisione
- Numeri adottano le specifiche IEEE 754 che indicano anche
  - Infinito positivo e negativo (`Float.POSITIVE_INFINITY`, `Float.NEGATIVE_INFINITY` stessa cosa per double)
  - NaN ovvero not a number (`Float.NaN`)

Tipo	Spazio Memoria	Range
`float`	`4 byte`	`±3,40282347E+38F` (6 o 7 cifre decimali significative)
`double`	`8 byte`	`±1,79769313486231570E+308` (15 cifre decimali significative)

Tipi Reference

- Costituiscono dei *puntatori* (reference) a degli oggetti
- Tre tipi
  - Classe
    - `Point p`
  - Interfaccia
    - `Comparable c1`
    - `Comparator c2`
  - Array
    - `int[] a1`
    - `Point[] p1`

Variabile (1)

- Una variabile è un'area di memoria ed ha associato un tipo che può essere di tipi *primitivo* o *reference*
- Viene specificata mediante l'**identificatore**
- Una variabile contiene sempre un valore che è compatibile con il suo tipo
- Sintassi

```
[modificatori] tipo_di_dato nome_della_variabile [= inizializzazione];
```

- Tipi
  - Variabile di istanza
  - Variabile di classe (`static`)
  - **Variabile locale**
  - Componenti dell'array
  - Parametri dei metodi e dei costruttori
  - Parametro di un exception-handler: `catch(Exception e)`
  - Variabili final (`final`)

Variabile (2)

- Variabile locale deve essere **esplicitamente** inizializzata prima di essere utilizzata, ovvero deve trovarsi alla **sinistra** dell'operatore di assegnamento
- Esempi

1)

```
{
        int k;
        while (true) {
                k = n;
                if (k >= 5) break;  //OK
                n = 6;
        }
        System.out.println(k);
}
```

Variabile (3)

2)

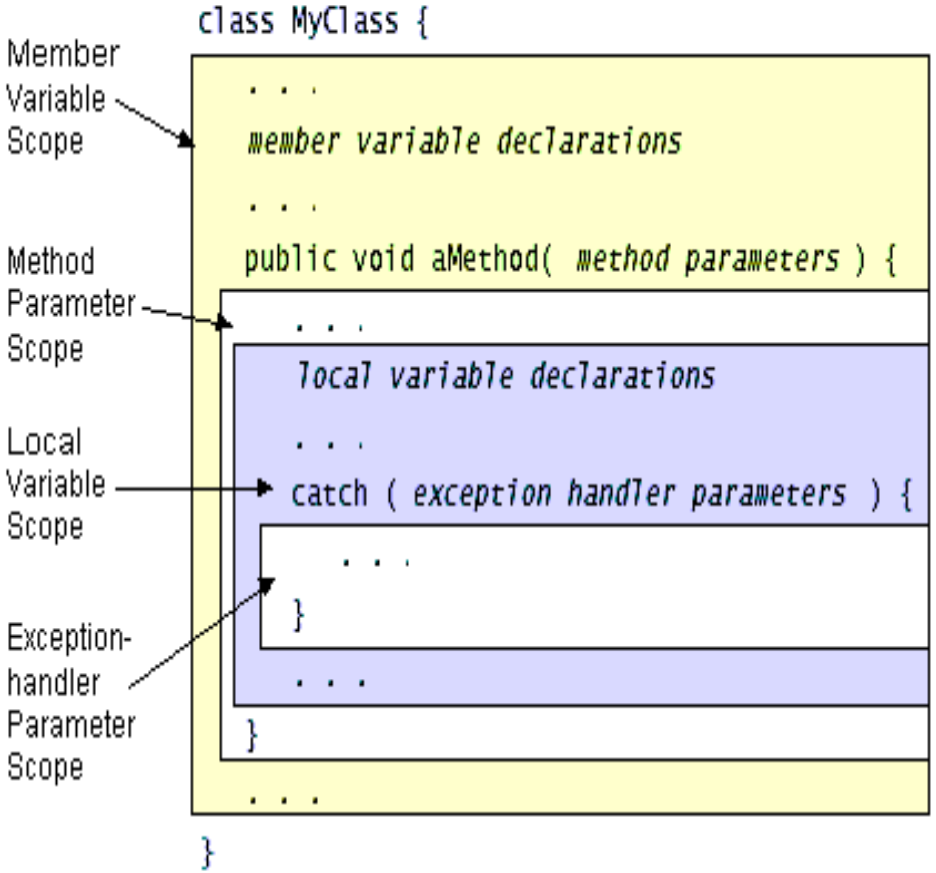
```
{
  int k;
  while (n < 4) {
          k = n;
          if (k >= 5) break;
          n = 6;
  }
  System.out.println(k); //NOT OK
}
```

3)

```
{
  int k;
  int n = 5;
  if (n > 2)
     k = 3;
  System.out.println(k); //NOT OK
}
```

Scope (1)

- Regione del programma all'interno del quale la variabile può essere riferita utilizzando il suo nome
- Determina quando il sistema crea e distrugge le memoria necessaria a contenere la variabile
- **Scope ◇ visibilità** (modificatori di accesso)



Scope (2)

- Variabili locali
  - Scope è dalla dichiarazione della variabile locale fino alla fine del blocco che la racchiude

```
public static void main(String args[]) {
        Punto punto1;
        punto1 = new Punto();
        punto1.x = 2;
        punto1.y = 6;
        if (args[0].equals("punto")) {
            Punto punto2 = new Punto();
            punto2.x = 0;
            punto2.y = 1;
        }
        System.out.println(punto1.x);
        System.out.println(punto1.y);
        System.out.println(punto2.x); //Errore in compilazione
        System.out.println(punto2.y); //Errore in compilazione
}
```

Scope (3)

- Parametri formali
  - Scope è il corpo del metodo dove è stato dichiarato il parametro formale
- Esempio

```
int setX(int x) {          // x is a method parameter
      int oldx = x;    // oldx is a local variable
      return oldx;
}
```

Rappresentazione dati (1)

- JVM definisce diverse aree di memoria che memorizzano i dati durante l'esecuzione di un programma
- Alcune di tali aree sono create allo start-up e sono distrutte al termine dell'esecuzione della VM
- Altre sono associate ai singoli thread
- Diversi tipi
  - Stack
  - Heap
  - Area metodi
  - ...

Rappresentazione dati (2): Stack

- VM associa ad ogni thread uno stack
- Pila (LIFO) che memorizza i cosiddetti *Frames*
- Frame
  - Viene creato ogni volta che viene invocato un metodo
  - Viene distrutto al termine dell'esecuzione del metodo
  - Contiene
    - Valori delle variabili locali
    - Valori dei parametri formali
    - ....

Rappresentazione dati (3): Heap

- Condiviso tra tutti i thread della VM
- Contiene tutti le istanze delle classi e gli array che sono allocati
- E' creato allo start-up della VM
- Garbage Collector è un thread a bassa priorità che elimina gli oggetti non più utilizzati
- Può avere una dimensione fissa oppure essere espanso o diminuito quando necessario

Rappresentazione dati (4): Area Metodi

- Condivisa tra tutti i thread della VM
- Memorizza strutture associate alla classe
  - Runtime constant pool
  - Codice dei metodi e costruttori
  - Codice per inizializzatori statici e di classe
  - **Variabili statiche**
- Logicamente fa parte dell'heap
- La gestione e l'organizzazione è demandata alle singole implementazioni

Rappresentazione dati (5)

```
public class Point {
      private int x = 10
      private int y = 20;

      public Point(int dx, int dy) {
            x = dx;
            y = dy;
      }
}
```

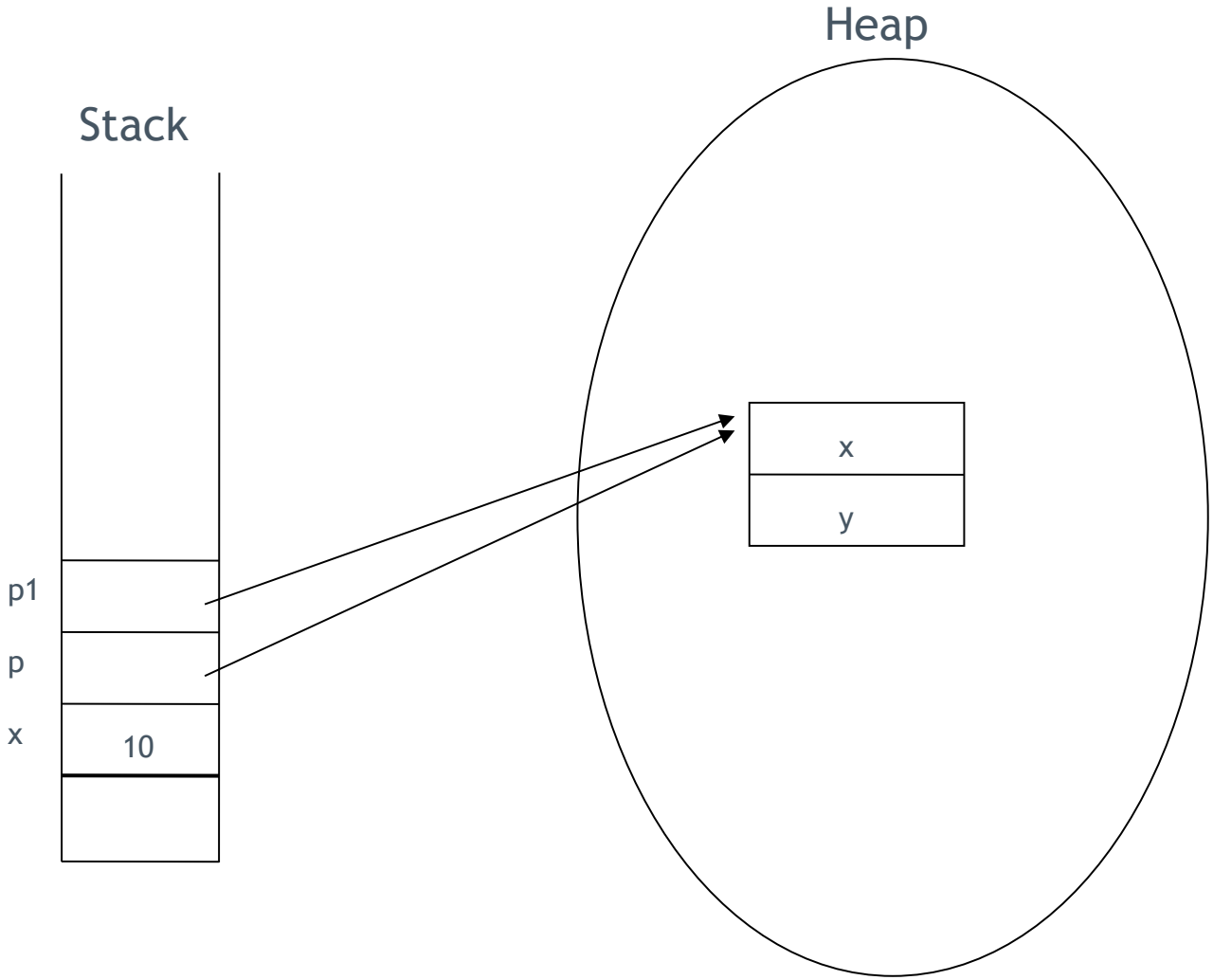
Rappresentazione dati (6)

```
public class Test {
      public static void main(String[] args) {
            int x = 10;
            Point p = new Point( 100, 200 );
            Point p1;
            p1 = p;
      }
}
```

Rappresentazione dati (7)

- Passi eseguiti dalla VM durante la creazione di un oggetto (`new Point(100, 200)`)
  - Viene allocato dello spazio nell'heap per contenere l'oggetto
  - Variabili di istanza vengono inizializzate al loro valore di default
  - Vengono eseguite le esplicite inizializzazioni
  - Viene eseguito il costruttore
  - Variabile viene assegnata all'oggetto

Rappresentazione dati (8)



Passaggio per valore (1)

- Java ammette soltanto il passaggio per valore dei parametri nei metodi e costruttori
- Nell'invocazione del metodo viene effettuata una copia (all'interno del frame del metodo nello stack) del valore contenuto nel parametro attuale nel parametro formale
- Modifiche del valore del parametro formale all'interno del metodo non hanno effetto nel parametro attuale
- Nota
  - Nel caso di tipi **reference** viene copiato il riferimento →
    - valori contenuti nell'oggetto possono essere modificati

Passaggio per valore (2)

```
public static void tripleValue(double x) {

      x = 3 * x;

}

double percent = 10;
tripleValue(percent);
```

Passaggio per valore (3)

```
public class Swap {
        public static void main(String[] args) {
                int x = 100;
                int y = 200;
                System.out.println("x=" + x);
                System.out.println("y=" + y);
                swap( x, y );
                System.out.println("x=" + x);
                System.out.println("y=" + y);
        }
        public static void swap(int x, int y) {
                int temp = x;
                x = y;
                y = temp;
        }
}
```

Passaggio per valore (4)

```
public class Swap {

   public static void main(String[] args) {
    Point p1 = new Point(100,200);
    Point p2 = new Point(300,400);
    System.out.println("p1=" + p1);
    System.out.println("p2=" + p2);
    swap( p1, p2 );
    System.out.println("p1=" + p1);
    System.out.println("p2=" + p2);

   }

   public static void swap(Point p1, Point p2) {
    Point temp = p1;
    p1 = p2;
    p2 = temp;
   }
}
```

Passaggio per valore (5)

```
public class Swap {
            public static void main(String[] args) {
                        Point p1 = new Point(100,200);
                        Point p2 = new Point(300,400);
                        System.out.println("p1=" + p1);
                        System.out.println("p2=" + p2);
                        swap( p1, p2 );
                        System.out.println("p1=" + p1);
                        System.out.println("p2=" + p2);
            }
            public static void swap(Point p1, Point p2) {
                        int tempx = p1.x;
                        int tempy = p1.y;
                        p1.x = p2.x;
                        p1.y = p2.y;
                        p2.x = tempx;
                        p2.y = tempy;
            }
}
```