

Corso Spring Web MVC

Emanuele Galli

www.linkedin.com/in/egalli/

Java

Linguaggio di programmazione general-purpose, multi-platform, network-centric, class-based, object-oriented progettato da James Gosling @ Sun Microsystems.

- JVM: Java Virtual Machine
- JRE: Java Runtime Environment
- JDK: Java Development Kit

Versioni

- 23 maggio 1995: prima release
- 1998 1.2 (J2SE)
- 2004 1.5 (J2SE 5.0)
- 2011 Java SE 7
- 2014 Java SE 8 (LTS)
- 2019 Java SE 12

Link utili

The Java Language Specifications

<https://docs.oracle.com/javase/specs/>

Java Platform, Standard Edition Documentation

<https://docs.oracle.com/en/java/javase/index.html>

Java SE 8 API Specification

<https://docs.oracle.com/javase/8/docs/api/index.html>

The Java Tutorials

<https://docs.oracle.com/en/java/javase/index.html>

Say hello /1

```
// Hello.java
```

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

Say hello /2

- JDK (8) from Oracle (for Windows x64 or ...)

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

- javac: Hello.java → Hello.class

source code → bytecode

- java: Hello.class → "Hello!"

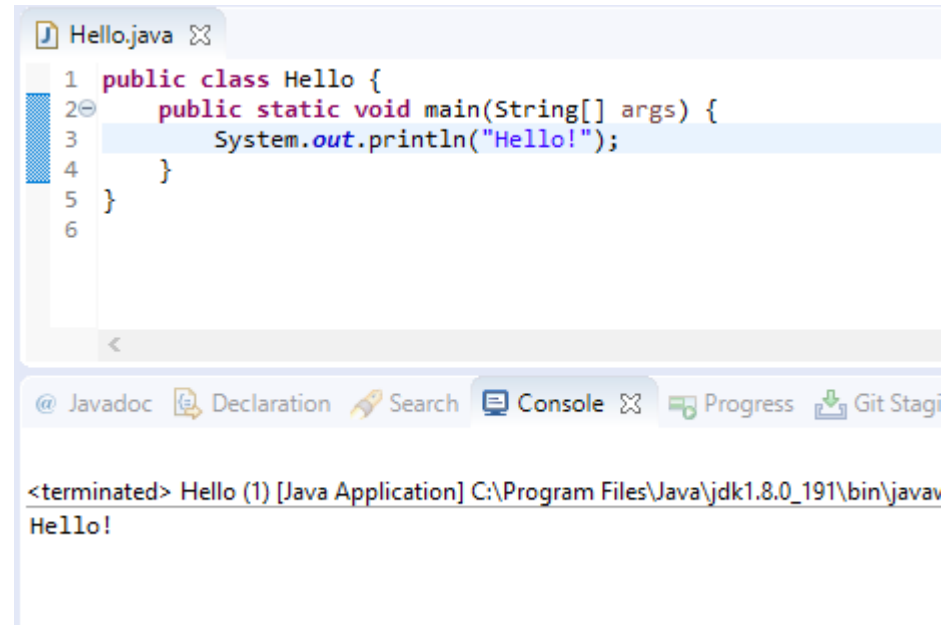
bytecode → machine code

Say hello /3

Integrated Development Environment (IDE)

- IntelliJ IDEA
- Eclipse IDE ← <https://www.eclipse.org/downloads/>
- Apache NetBeans
- ...

Hello!



```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.println("Hello!");  
4     }  
5 }  
6
```

<terminated> Hello (1) [Java Application] C:\Program Files\Java\jdk1.8.0_191\bin\javav
Hello!

Struttura del codice /1

- Dichiarazioni
 - **Package** (collezione di classi)
 - **Import** (accesso a classi di altri package)
 - **Class** (solo una “public” per file)
- Commenti
 - **Multi-line**
 - **Single-line**
 - **Javadoc-style**

```
/*  
 * A simple Pi.java source file  
 */  
package dd.hello;  
  
import java.lang.Math; // not required  
  
/**  
 * @author manny  
 */  
public class Pi {  
    public static void main(String[] args) {  
        System.out.println(Math.PI);  
    }  
}  
  
class PackageClass {  
    // TBD  
}
```

Struttura del codice /2

- Metodi
 - **main** (definito)
 - **println** (invocato)
- Parentesi
 - **Graffe** (blocchi, body di classi e metodi)
 - **Tonde** (liste parametri di metodi)
 - **Quadre** (array)
- **Statement** (sempre terminati da punto e virgola!)

```
/*  
 * A simple Pi.java source file  
 */  
package dd.hello;  
  
import java.lang.Math; // not required  
  
/**  
 * @author manny  
 */  
public class Pi {  
    public static void main(String[] args) {  
        System.out.println(Math.PI);  
    }  
}  
  
class PackageClass {  
    // TBD  
}
```

Variabili e tipi di dato

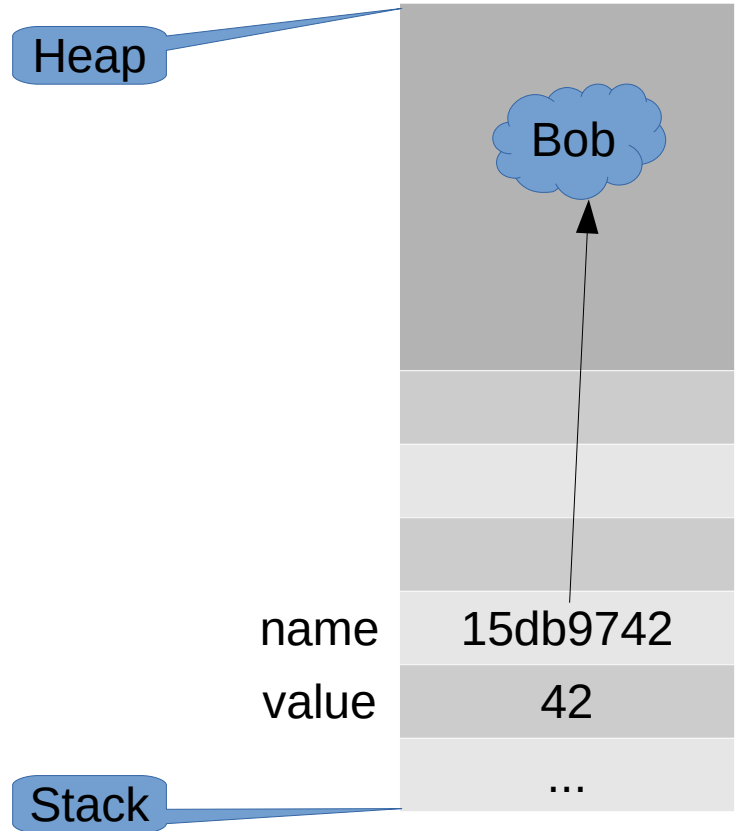
- Variabile: una locazione di memoria con un nome usato per accederla.
- Tipi di dato: determina valore della variabile e operazioni disponibili.
 - Primitive data type
 - Reference data type (class / interface)

Tipi primitivi

bit			signed integer	floating point IEEE 754
1(?)	boolean	false true		
8			byte	
16	char	'\u0000' '\uFFFF'	short	
32			int	float
64			long	double

Primitivi vs Reference

```
// primitivo  
int value = 42;  
  
// reference  
String name = "Bob";
```



String

- Reference type che rappresenta una sequenza immutabile di caratteri
- StringBuilder, controparte mutabile, per creare stringhe complesse

```
String s = new String("hello");
```

Forma standard

```
String t = "hello";
```

Forma semplificata equivalente

Operatori unari

++ incremento

-- decremento

prefisso: “naturale”

postfisso: ritorna il valore
prima dell'operazione

+ positivo (useless)

- cambia il segno

```
int value = 1;
System.out.println(value);           // 1
System.out.println(++value);        // 2
System.out.println(--value);         // 1
System.out.println(value++);        // 1
System.out.println(value);          // 2
System.out.println(value--);        // 2
System.out.println(value);          // 1
System.out.println(+value);         // 1
System.out.println(-value);         // -1
```

Operatori aritmetici

+ addizione

- sottrazione

* moltiplicazione

/ divisione (intera)

% modulo

```
int a = 10;  
int b = 3;  
  
System.out.println(a + b); // 13  
System.out.println(a - b); // 7  
System.out.println(a * b); // 30  
System.out.println(a / b); // 3  
System.out.println(a % b); // 1
```


Concatenazione di stringhe

- L'operatore + è overloaded per le stringhe.
- Se un operando è di tipo stringa, l'altro viene convertito a stringa e si opera la concatenazione.

```
System.out.println("Resistence" + " is " + "useless" );  
System.out.println("Solution: " + 42 );
```

Operatori relazionali

<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
==	Uguale
!=	Diverso

```
int alpha = 12;  
int beta = 21;  
int gamma = 12;
```

```
System.out.println("alpha < beta? " + (alpha < beta)); // true  
System.out.println("alpha < gamma? " + (alpha < gamma)); // false  
System.out.println("alpha <= gamma? " + (alpha <= gamma)); // true  
  
System.out.println("alpha > beta? " + (alpha > beta)); // false  
System.out.println("alpha > gamma? " + (alpha > gamma)); // false  
System.out.println("alpha >= gamma? " + (alpha >= gamma)); // true  
  
System.out.println("alpha == beta? " + (alpha == beta)); // false  
System.out.println("alpha == gamma? " + (alpha == gamma)); // true  
  
System.out.println("alpha != beta? " + (alpha != beta)); // true  
System.out.println("alpha != gamma? " + (alpha != gamma)); // false
```

Operatori logici (e bitwise)

&&	AND
	OR
!	NOT
&	AND
	OR
^	XOR

```
boolean alpha = true;
boolean beta = false;

System.out.println(alpha && beta);    // false
System.out.println(alpha || beta);    // true
System.out.println(!alpha);           // false
System.out.println(alpha & beta);      // false
System.out.println(alpha | beta);      // true

int gamma = 0b101;    // 5
int delta = 0b110;    // 6

System.out.println(gamma & delta);     // 4 = 0100
System.out.println(gamma | delta);     // 7 = 0111
System.out.println(gamma ^ delta);     // 3 = 0011
```

Operatori di assegnamento

=	Assegnamento
+=	Aggiungi e assegna
-=	Sottrai e assegna
*=	Moltiplica e assegna
/=	Dividi e assegna
%=	Modulo e assegna
&=	AND e assegna
=	OR e assegna
^=	XOR e assegna

```
int alpha = 2;
```

```
alpha += 8;           // 10
```

```
alpha -= 3;           // 7
```

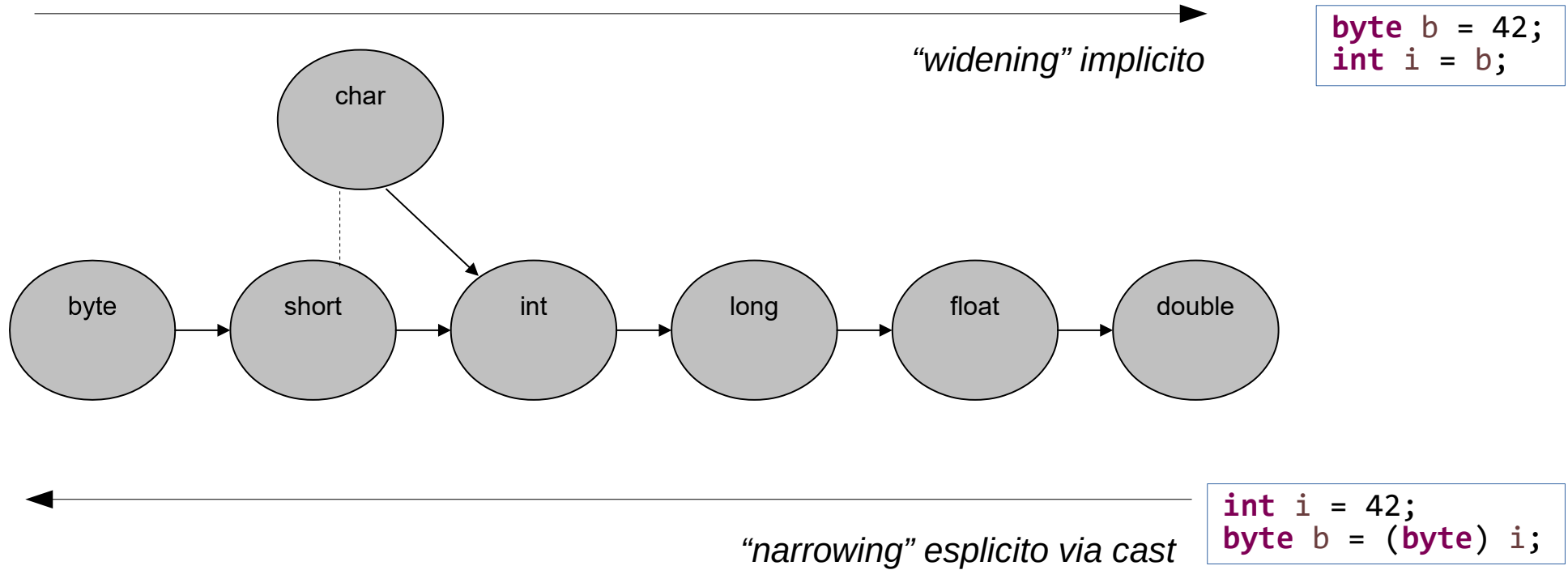
```
alpha *= 2;           // 14
```

```
alpha /= 2;           // 7
```

```
alpha %= 5;           // 2
```

```
System.out.println(alpha);
```

Cast tra primitivi



Array

- Sequenza di “length” valori dello stesso tipo, memorizzati nello heap.
- Accesso per indice, a partire da 0.

```
int[] array = new int[12];  
array[0] = 7;  
  
int value = array[5];
```

```
int[] array = { 1, 4, 3 };  
  
if(array.length != 3) {  
    System.out.println("Unexpected");  
}
```

```
int[][] array2d = new int[4][5];  
  
int value = array2d[2][3];
```

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]

if ... else if ... else

- Se la condizione è vera, si esegue il blocco associato.
- Altrimenti, se presente si esegue il blocco “else”.

```
if (condition) {  
    doSomething();  
}  
nextStep();
```

```
if (condition) {  
    doSomething();  
} else {  
    doSomethingElse();  
}  
nextStep();
```

```
if (condition) {  
    doSomething();  
} else if (otherCondition) {  
    doSomethingElse();  
} else {  
    doSomethingDifferent();  
}  
nextStep();
```

switch

Scelta multipla su byte, short, char, int, String, enum

```
int value = 1;

// ...

switch (value) {
case 1:
    f();
    break;
case 2:
    g();
    break;
default:
    h();
    break;
}
```

```
String value = "1";

// ...

switch (value) {
case "1":
    f();
    break;
case "2":
    g();
    break;
default:
    h();
    break;
}
```

```
public enum WeekendDay {
    SATURDAY, SUNDAY
}

WeekendDay day = WeekendDay.SATURDAY;

// ...

switch (day) {
case SATURDAY:
    f();
    break;
case SUNDAY:
    g();
    break;
}
```


loop

```
while (condition) {  
    // ...  
    if (something) {  
        condition = false;  
    }  
}
```

```
for (int i = 0; i < 10; i++) {  
    // ...  
    if (i == 4) {  
        continue;  
    }  
    // ...  
}
```

```
for (;;) {  
    // ...  
    if (something) {  
        break;  
    }  
    // ...  
}
```

```
do {  
    // ...  
    if (something) {  
        condition = false;  
    }  
} while (condition);
```

```
int[] array = new int[12];  
for (int value : array) {  
    System.out.println(value);  
}
```

Classi e oggetti

- Classe:
 - Ogni classe è definita in un package
 - Descrive un nuovo tipo di dati, con proprietà e metodi
- Oggetto
 - Istanza di una classe, suo modello di riferimento

Reference a MyClass

Crea un oggetto MyClass

```
MyClass object = new MyClass();
```

Metodo

- Blocco di codice associato ad un oggetto (default) o a una classe (static)
- Identificato da:
 - return type
 - nome
 - lista dei parametri

```
class MyClass {  
    static String h() {  
        return "Hi";  
    }  
    int f(int a, int b) {  
        // ...  
        return a * b;  
    }  
    void g(boolean flag) {  
        if (flag) {  
            System.out.println("Hello!");  
            return;  
        }  
        // ...  
        System.out.println("Goodbye");  
    }  
}
```

Constructor (ctor)

- Metodo speciale, ha lo stesso nome della classe, è invocato durante la creazione di un oggetto via “new” per inizializzarne lo stato
- Non ha return type (nemmeno void)
- Se una classe non ha ctor, Java ne crea uno di default senza parametri (che non fa niente)

Alcuni metodi di String

- char charAt(int)
 - int compareTo(String)
 - String concat(String)
 - boolean contains(CharSequence)
 - boolean equals(Object)
 - int indexOf(int)
 - int indexOf(String)
 - boolean isEmpty()
 - int lastIndexOf(int ch)
 - int length()
 - String replace(char, char)
 - String[] split(String)
 - String substring(int)
 - String toLowerCase()
 - String toUpperCase()
 - String trim()
- Tra i metodi statici:
- String format(String, Object...)
 - String join(CharSequence, CharSequence...)
 - String valueOf(Object)

Alcuni metodi di StringBuilder

- `StringBuilder(int)`
- `StringBuilder(String)`
- `StringBuilder append(Object)`
- `char charAt(int)`
- `StringBuilder delete(int, int)`
- `void ensureCapacity(int)`
- `int indexOf(String)`
- `StringBuilder insert(int, Object)`
- `int length()`
- `StringBuilder replace(int, int, String)`
- `StringBuilder reverse()`
- `void setCharAt(int, char)`
- `void setLength(int)`
- `String toString()`

La classe Math

Proprietà statiche

- E – base del logaritmo naturale
- PI – pi greco

Alcuni metodi statici

- double abs(double) // int, ...
- int addExact(int, int) // multiply ...
- double ceil(double)
- double cos(double) // sin(), tan()
- double exp(double)
- double floor(double)
- double log(double)

... alcuni metodi statici

- double max(double, double) // int, ...
- double min(double, double) // int, ...
- double pow(double, double)
- double random()
- long round(double)
- double sqrt(double)
- double toDegrees(double) // approx
- double toRadians(double) // approx

Tre principi OOP

- Incapsulamento per mezzo di classi
 - Visibilità pubblica / privata
- Ereditarietà in gerarchie di classi
 - Dal generale al particolare
- Polimorfismo
 - Una interfaccia, molti metodi

Access modifier per data member

- Aiutano l'incapsulamento
 - Privato
 - Protetto (ereditarietà)
- Normalmente sconsigliati
 - Package (default)
 - Pubblico

```
package my.package.example;

public class Sample {
    private int a;
    protected short b;
    static double c;
    // public long d;

    static {
        c = 18;
    }

    public Sample() {
        this.a = 42;
        this.b = 23;
    }

    // ...
}
```

Static initializer

Costruttore

Access modifier per metodi

- Pubblico
- Package (usi speciali)
- Protetto / Privato (helper)

```
package my.package.example;

public class Sample {
    // ...

    static private double f() {
        return c;
    }

    void g() {
        f();
    }

    public int h() {
        return a / 2;
    }
}
```

Lo “scope” delle variabili

- Locali
- Member (field, property)
 - instance (default)
 - static

```
public class Variables {  
    private static int staticMember = 5;  
    private int member = 5;  
  
    public void f() {  
        long local = 7;  
        if (staticMember == 2) {  
            short inner = 12;  
            staticMember = 1 + inner;  
            member = 3 + local;  
        }  
    }  
  
    public static void main(String[] args) {  
        double local = 5;  
        System.out.println(local);  
        staticMember = 12;  
    }  
}
```

Inizializzazione delle variabili

- Esplicita per assegnamento (preferita)
 - primitivi: diretto
 - reference: via new
- Implicita by default (solo member)
 - primitivi
 - numerici: 0
 - boolean: false
 - reference: null

```
int i = 42;  
String s = new String("Hello");
```

```
int i;           // 0  
boolean flag;    // false  
String s;        // null
```

Tipi wrapper

- Controparte reference dei tipi primitivi
 - Boolean, Character, Byte, Short, Integer, Float, Double
- Boxing esplicito
 - Costruttore
 - Static factory method (preferito)
- Unboxing esplicito
 - Metodi definiti nel wrapper
- Auto-boxing
- Auto-unboxing

```
Integer i = new Integer(1);  
Integer j = Integer.valueOf(2);  
  
int k = j.intValue();  
  
Integer m = 3;  
  
int n = k;
```

Interfaccia

- Cosa deve fare una classe, non *come* deve farlo (fino a Java 8)
- Una classe “implements” una interfaccia
- Un’interfaccia “extends” un’altra interfaccia
- I metodi sono (implicitamente) public
- Le eventuali proprietà sono static final

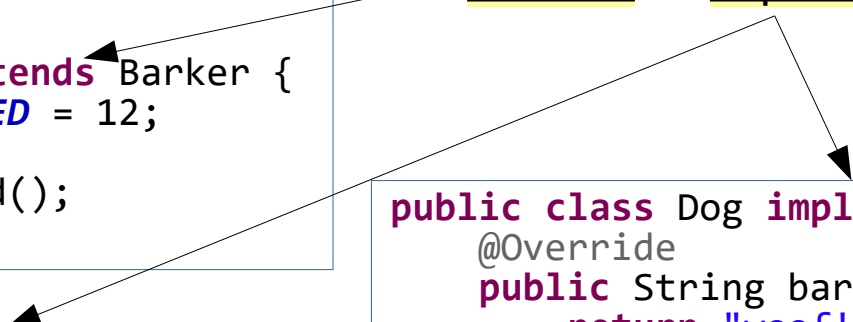
Interfacce e classi

```
interface Barker {  
    String bark();  
}
```

```
interface BarkAndWag extends Barker {  
    int AVG_WAGGING_SPEED = 12;  
  
    int tailWaggingSpeed();  
}
```

```
public class Fox implements Barker {  
    @Override  
    public String bark() {  
        return "yap!";  
    }  
}
```

extends vs implements



```
public class Dog implements BarkAndWag {  
    @Override  
    public String bark() {  
        return "woof!";  
    }  
  
    @Override  
    public int tailWaggingSpeed() {  
        return BarkAndWag.AVG_WAGGING_SPEED;  
    }  
}
```

abstract class

- Una classe abstract non può essere istanziata
- Un metodo abstract non ha body
- Una classe che ha un metodo abstract deve essere abstract, ma non viceversa
- Una subclass di una classe abstract o implementa tutti i suoi metodi abstract o è a sua volta abstract

Ereditarietà

- extends (is-a)
 - Subclasse che estende una già esistente
 - Eredita proprietà e metodi della superclass
 - p.es.: Mammal superclass di Cat e Dog
- Aggregazione (has-a)
 - Classe che ha come proprietà un'istanza di un'altra classe
 - p.es: Tail in Cat e Dog

Ereditarietà in Java

- Single inheritance: una sola superclass
- Implicita derivazione da Object (che non ha superclass) by default
- Una subclass può essere usata al posto della sua superclass (is-a)
- Una subclass può aggiungere proprietà e metodi a quelli ereditati dalla superclass (attenzione a non nascondere proprietà della superclass con lo stesso nome!)
- Costruttori e quanto nella parte private della superclass non è ereditato dalla subclass
- Subclass transitivity: C subclass B, B subclass A \rightarrow C subclass A

this vs super

- Reference all'oggetto corrente
 - this, come istanza della classe
 - super, come istanza della superclass
- ctor → ctor: (primo statement)
 - this() – nella classe
 - super() – nella superclass

Esempio di ereditarietà

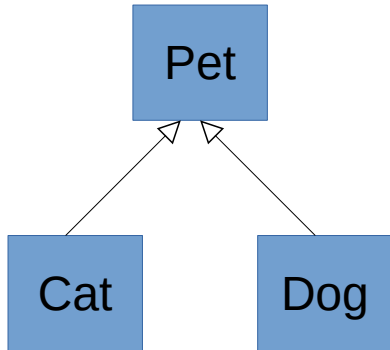
```
public class Pet {  
    private String name;  
  
    public Pet(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

```
Dog tom = new Dog("Tom", 2.42);  
  
// ...  
  
String name = tom.getName();  
double speed = tom.getSpeed();
```

```
public class Dog extends Pet {  
    private double speed;  
  
    public Dog(String name) {  
        this(name, 0);  
    }  
  
    public Dog(String name, double speed) {  
        super(name);  
        this.speed = speed;  
    }  
  
    public double getSpeed() {  
        return speed;  
    }  
}
```

Reference casting

- Upcast: da subclass a superclass (sicuro)
- Downcast: da superclass a subclass (!?)
 - Protetto con l'uso di **instanceof**



```
// Cat cat = (Cat) new Dog(); // Cannot cast from Dog to Cat

Pet pet = new Dog();
Dog dog = (Dog) pet; // OK
Cat cat = (Cat) pet; // trouble at runtime
if(pet instanceof Cat) {
    Cat tom = (Cat) pet;
}
```

Eccezioni

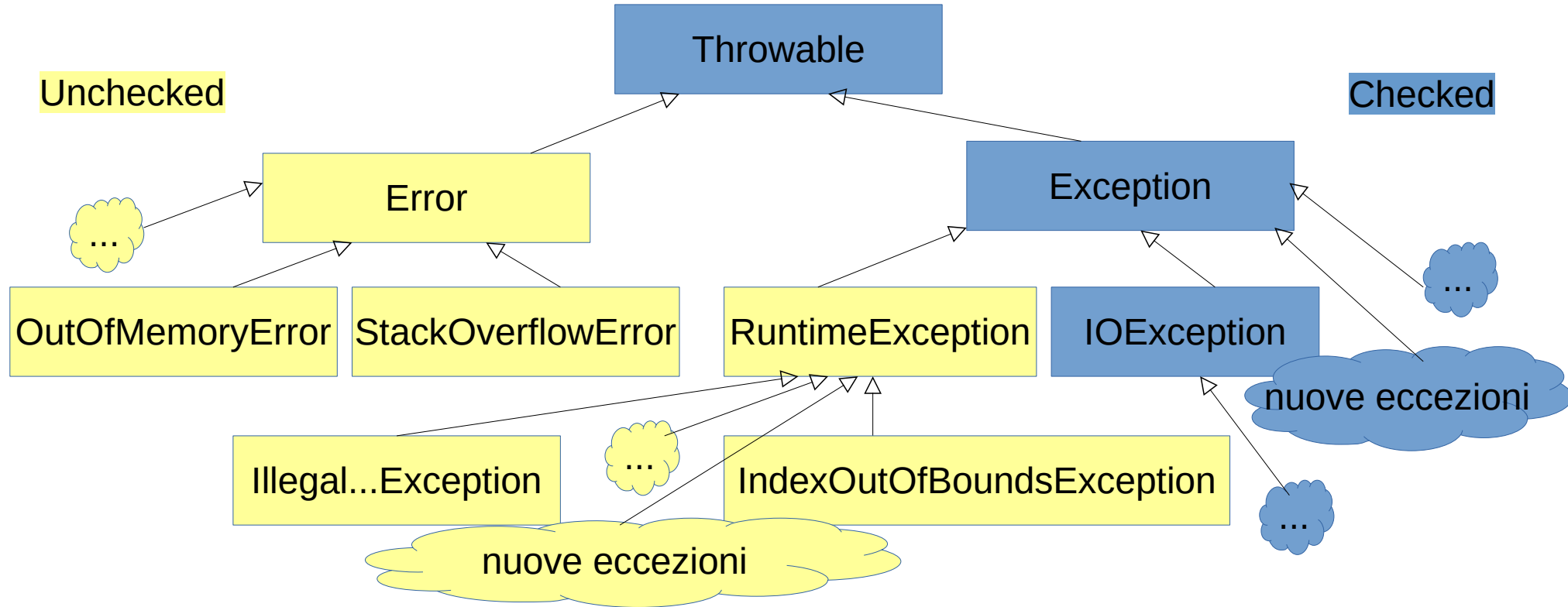
- Obbligano il chiamante a gestire gli errori
 - Unhandled exception → terminazione del programma
- Evidenziano il flusso normale di esecuzione
- Semplificano il debug esplicitando lo stack trace
- Possono chiarire il motivo scatenante dell'errore
- Checked vs unchecked

try – catch – finally

- try: esecuzione protetta
- catch: gestisce uno o più possibili eccezioni
- finally: sempre eseguito, alla fine del try o dell'eventuale catch
- Ad un blocco try deve seguire almeno un blocco catch o finally
- “throws” nella signature, “throw” per “tirare” una eccezione.

```
void f() {  
    try {  
        g();  
    } catch (Exception ex) {  
        // ...  
    } finally {  
        cleanup();  
    }  
}  
  
// ...  
  
void g() throws Exception {  
    // ...  
    if (somethingUnexpected()) {  
        throw new Exception();  
    }  
}
```

Gerarchia delle eccezioni

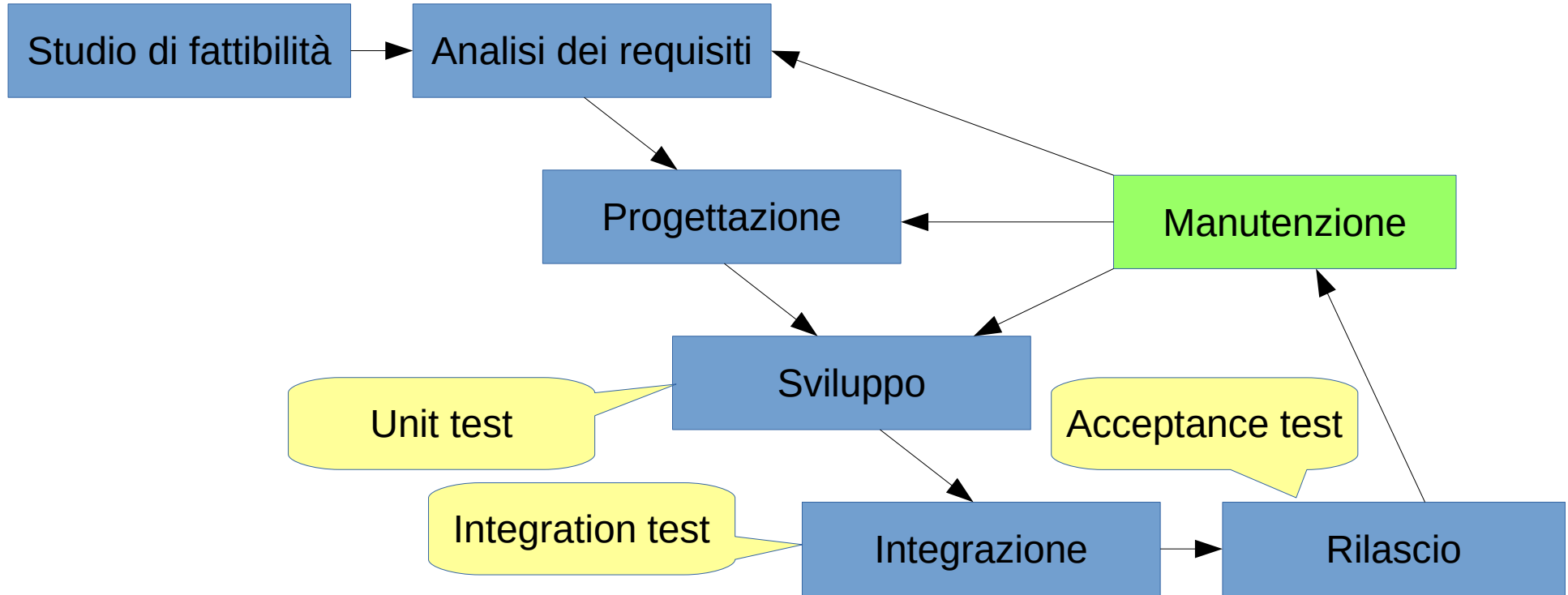


Ciclo di vita del software

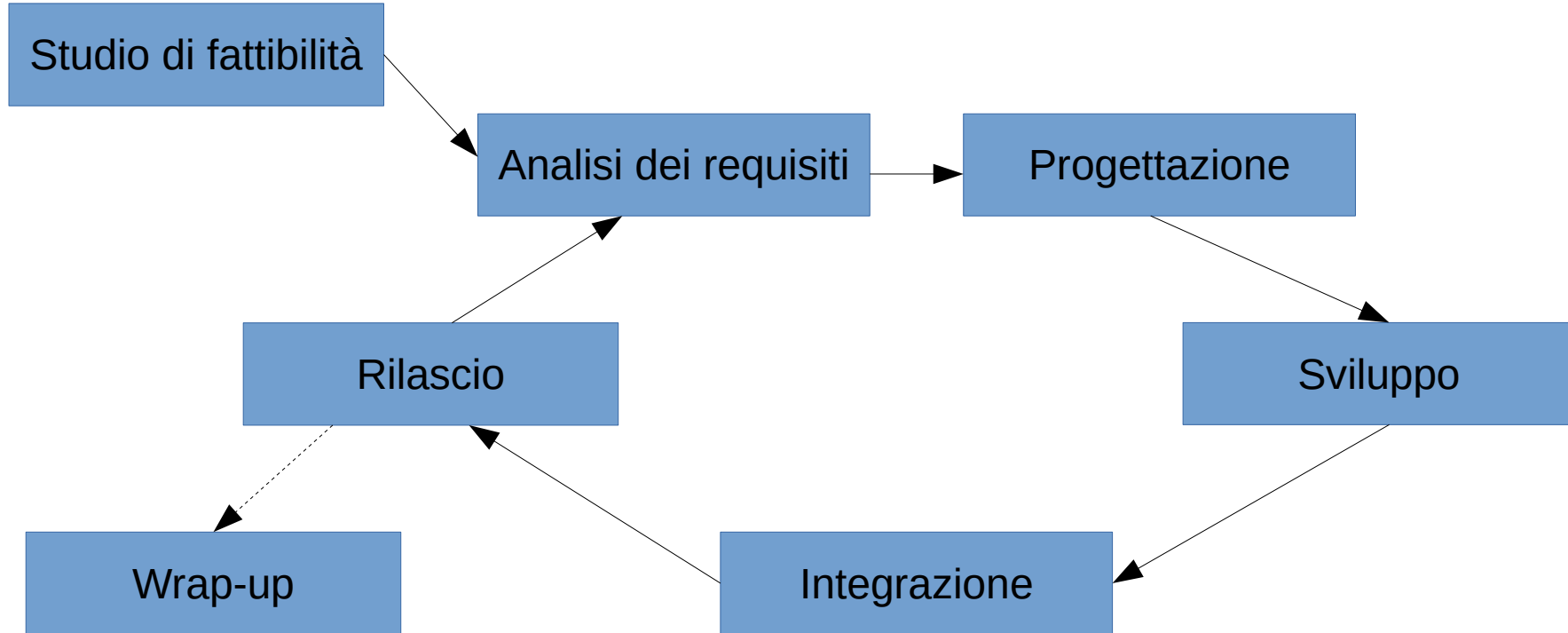
Come gestire la complessità di un progetto?

- Divide et impera
- Struttura
- Documentazione
- Milestones
- Comunicazione e interazione tra partecipanti

Modello a cascata (waterfall)



Modello agile



Input con Scanner

- Legge input formattato e lo converte in formato binario
- Può leggere da `InputStream`, `File`, `String`, o altre classi che implementano `Readable` o `ReadableByteChannel`
- Uso generale di `Scanner`:
 - Il ctor associa l'oggetto scanner allo stream in lettura
 - Loop su `hasNext...()` per determinare se c'è un token in lettura del tipo atteso
 - Con `next...()` si legge il token
 - Terminato l'uso, ricordarsi di invocare `close()` sullo scanner

Un esempio per Scanner

```
import java.util.Scanner;

public class Sample {
    public static void main(String[] args) {
        System.out.println("Please, enter a few numbers");
        double result = 0;

        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNext()) {
            if (scanner.hasNextDouble()) {
                result += scanner.nextDouble();
            } else {
                System.out.println("Bad input, discarded: " + scanner.next());
            }
        }
        scanner.close(); // see try-with-resources
        System.out.println("Total is " + result);
    }
}
```

try-with-resources

Per classi che implementano AutoCloseable

```
double result = 0;

// try-with-resources
try(Scanner scanner = new Scanner(System.in)) {
    while (scanner.hasNext()) {
        if (scanner.hasNextDouble()) {
            result += scanner.nextDouble();
        } else {
            System.out.println("Bad input, discarded: " + scanner.next());
        }
    }
}

System.out.println("Total is " + result);
```

Java Util Logging

```
public static void someLog() {  
    Logger log =  
        Logger.getLogger("sample");  
  
    log.finest("finest message");  
    log.finer("finer message");  
    log.fine("fine message");  
    log.config("config message");  
    log.info("info message");  
    log.warning("warning message");  
    log.severe("severe message");  
}
```

```
public static void main(String[] args) {  
    Locale.setDefault(new Locale("en", "EN"));  
    Logger log = Logger.getLogger("sample");  
  
    someLog();  
  
    ConsoleHandler handler = new ConsoleHandler();  
    handler.setLevel(Level.ALL);  
    log.setLevel(Level.ALL);  
    log.addHandler(handler);  
    log.setUseParentHandlers(false);  
  
    someLog();  
}
```

Generic

- Supporto ad algoritmi generici che operano allo stesso modo su tipi differenti (es: collezioni)
- Migliora la type safety del codice
- In Java è implementato solo per references
- Il tipo (o tipi) utilizzato dal generic è indicato tra parentesi angolari (minore, maggiore)

Inner class

- Nested class: classe definita all'interno di un'altra classe
- La nested class ha accesso a tutti i membri della classe in cui è definita
- È possibile definirla come locale ad un blocco
- Inner class: non-static nested class
- Utili (ad es.) per semplificare la gestione di eventi

Reflection

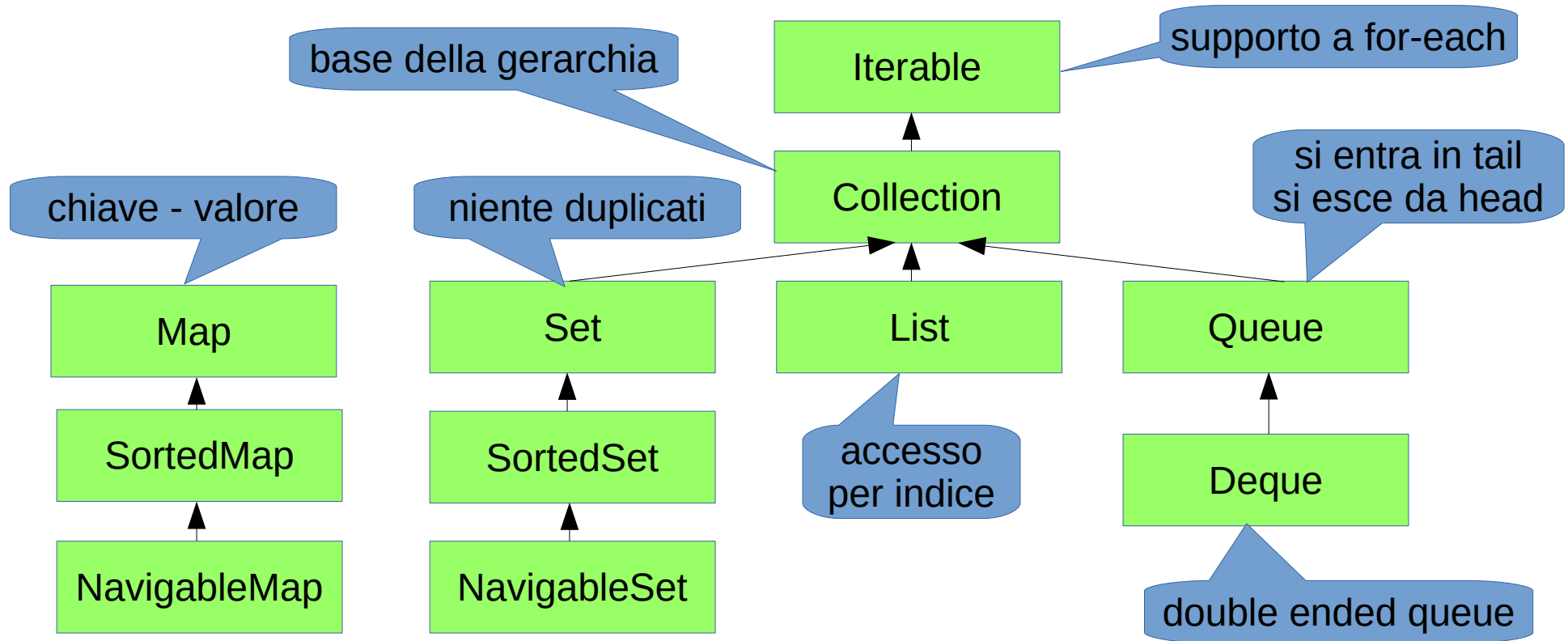
- Package `java.lang.reflect`
- Permette di ottenere a run time informazioni su di una classe
- “Class” è la classe che rappresenta una certa classe (!)
- Ad es: con `getMethods()` si possono ottenere tutti i metodi di una classe

```
Class<?> c = Integer.class;
Method[] methods = c.getMethods();
for(Method method: methods) {
    System.out.println(method);
}
```

Java Collections Framework

- Lo scopo è memorizzare e gestire gruppi di oggetti (solo reference, no primitive)
- Enfasi su efficienza, performance, interoperabilità, estensibilità, adattabilità
- Basate su alcune interfacce standard
- La classe Collections contiene algoritmi generici
- L'interfaccia Iterator dichiara un modo standard per accedere, uno alla volta, gli elementi di una collezione

Interfacce per Collection



Alcuni metodi in Collection<E>

- boolean add(E)
- boolean addAll(Collection<? extends E>)
- void clear()
- boolean contains(Object);
- boolean equals(Object);
- boolean isEmpty();
- Iterator<E> iterator();
- boolean remove(Object);
- boolean retainAll(Collection<?>);
- int size();
- Object[] toArray();
- <T> T[] toArray(T[]);

Alcuni metodi in List<E>

- void add(int, E)
- E get(int)
- int indexOf(Object)
- E remove(int)
- E set(int, E)

Alcuni metodi in SortedSet<E>

- E first()
- E last()
- SortedSet<E> subSet(E, E)

Alcuni metodi in NavigableSet<E>

- E ceiling(E), E floor(E)
- E higher(E), E lower(E)
- E pollFirst(), E pollLast()
- Iterator<E> descendingIterator()
- NavigableSet<E> descendingSet()

Alcuni metodi in Queue<E>

- boolean offer(E e)
- E element()
- E peek()
- E remove()
- E poll()

Alcuni metodi in Deque<E>

- void addFirst(E), void addLast(E)
- E getFirst(), E getLast()
- boolean offerFirst(E), boolean offerLast(E)
- E peekFirst(), E peekLast()
- E pollFirst(), E pollLast()
- E pop(), void push(E)
- E removeFirst(), E removeLast()

Alcuni metodi in Map<K, V>

Map.Entry<K,V>

- K getKey()
- V getValue()
- V setValue(V)

- void clear()
- boolean containsKey(Object)
- boolean containsValue(Object)
- Set<Map.Entry<K, V>> entrySet()
- V get(Object)
- V getOrDefault(Object, V)
- boolean isEmpty()
- Set<K> keySet()
- V put(K, V)
- V putIfAbsent(K, V)
- V remove(Object)
- boolean remove(Object, Object)
- V replace(K key, V value)
- int size()
- Collection<V> values()

Metodi in NavigableMap<K, V>

- Map.Entry<K,V> ceilingEntry(K)
- K ceilingKey(K)
- Map.Entry<K,V> firstEntry()
- Map.Entry<K,V> floorEntry(K)
- K floorKey(K)
- NavigableMap<K,V> headMap(K, boolean)
- Map.Entry<K,V> higherEntry(K)
- K higherKey(K key)
- Map.Entry<K,V> lastEntry()
- Map.Entry<K,V> lowerEntry(K)
- K lowerKey(K)
- NavigableSet<K> navigableKeySet()
- Map.Entry<K,V> pollFirstEntry()
- Map.Entry<K,V> pollLastEntry()
- SortedMap<K,V> subMap(K, K)
- NavigableMap<K,V> tailMap(K, boolean)

ArrayList<E>

- implements List<E>
- Array dinamico vs standard array (dimensione fissa)
- Ctors
 - ArrayList() // capacity = 10
 - ArrayList(int) // set capacity
 - ArrayList(Collection<? extends E>) // copy

LinkedList<E>

- implements List<E>, Deque<E>
- Lista doppiamente linkata
- Accesso diretto solo a head e tail
- Ctors
 - LinkedList() // vuota
 - LinkedList(Collection<? extends E>) // copy

HashSet<E>

- implements Set<E>
- Basata sull'ADT hash table, $O(1)$, nessun ordine
- Ctors:
 - HashSet() // vuota, capacity 16, load factor .75
 - HashSet(int) // capacity
 - HashSet(int, float) // capacity e load factor
 - HashSet(Collection<? extends E>) // copy

LinkedHashSet<E>

- extends HashSet<E>
- Permette di accedere ai suoi elementi in ordine di inserimento
- Ctors:
 - `LinkedHashSet()` // capacity 16, load factor .75
 - `LinkedHashSet(int)` // capacity
 - `LinkedHashSet(int, float)` // capacity, load factor
 - `LinkedHashSet(Collection<? extends E>)` // copy

TreeSet<E>

- implements NavigableSet<E>
- Basata sull'ADT albero → ordine, $O(\log(N))$
- Gli elementi inseriti devono implementare Comparable ed essere tutti mutualmente comparabili
- Ctors:
 - TreeSet() // vuoto, ordine naturale
 - TreeSet(Collection<? extends E>) // copy
 - TreeSet(Comparator<? super E>) // sort by comparator
 - TreeSet(SortedSet<E>) // copy + comparator

TreeSet e Comparator

ordine naturale

comparator

plain

reversed

Java 8 lambda

```
List<String> data = Arrays.asList("alpha", "beta", "gamma", "delta");

TreeSet<String> ts = new TreeSet<>(data);

class MyStringComparator implements Comparator<String> {
    public int compare(String s, String t) {
        return s.compareTo(t);
    }
}

MyStringComparator msc = new MyStringComparator();

TreeSet<String> ts2 = new TreeSet<>(msc);
ts2.addAll(data);

TreeSet<String> ts3 = new TreeSet<>(msc.reversed());
ts3.addAll(data);

TreeSet<String> ts4 = new TreeSet<>((s, t) -> t.compareTo(s));
ts4.addAll(data);
```

HashMap<K, V>

- implements Map<K,V>
- Basata sull'ADT hash table, O(1), nessun ordine
- Mappa una chiave K (unica) ad un valore V
- Ctors:
 - HashMap() // vuota, capacity 16, load factor .75
 - HashMap(int) // capacity
 - HashMap(int, float) // capacity e load factor
 - HashMap(Map<? extends K, ? extends V>) // copy

TreeMap<K,V>

- implements NavigableMap<K,V>
- Basata sull'ADT albero → ordine, $O(\log(N))$
- Gli elementi inseriti devono implementare Comparable ed essere tutti mutualmente comparabili
- Ctors:
 - TreeMap() // vuota, ordine naturale
 - TreeMap(Comparator<? super K>) // sort by comparator
 - TreeMap(Map<? extends K, ? extends V>) // copy
 - TreeMap(SortedMap<K, ? extends V>) // copy + comparator

Multithreading

- Multitasking process-based vs thread-based
- L'interfaccia Runnable dichiara il metodo run()
- La classe Thread:
 - Ctors per Runnable
 - In alternativa, si può estendere Thread e ridefinire run()
 - start() per iniziare l'esecuzione

synchronized

- Metodo: serializza su this
- Blocco: serializza su oggetto specificato

comunicazione tra thread

- wait()
- notify() / notifyAll()

Oracle Database

- Multi-model DBMS
 - Relazionale
 - MySQL, SQL Server, PostgreSQL, DB2
 - NoSQL
 - MongoDB (doc), ElasticSearch (doc), Redis (k-v)

Database Relazionale

- Collezione di informazioni correlate, organizzata in tabelle
- Dati memorizzati in righe e ordinati per colonne
- Tabelle memorizzate in uno schema del database, che viene associato ad un utente
- Relazioni tra tabelle: primary key (PK) → foreign key (FK)
- Un utente può avere il permesso di accedere tabelle di altri schemi
- SQL (Structured Query Language) è il linguaggio standard per l'accesso a database relazionali

Structured Query Language (SQL)

- DQL – Data Query Language
 - SELECT
- DML – Data Manipulation Language
 - INSERT, UPDATE, DELETE
- DDL – Data Definition Language
 - CREATE, ALTER, DROP, RENAME, TRUNCATE
- TC – Transaction Control
 - COMMIT, ROLLBACK, SAVEPOINT
- DCL – Data Control Language
 - GRANT, REVOKE

Attivazione HR via SQL Plus

- Connessione a Oracle, per utente e database
 - sqlplus user/password@host
 - sqlplus / as sysdba
- Da Oracle 12 va specificato il container (orclpdb, xepdb1, ...)
 - alter session set container = orclpdb;
- Stato del database corrente
 - select name, open_mode from v\$pdb;
 - alter database open
- alter user hr identified by hr account unlock;

Creazione utente “oved”

- Connesso con sqlplus come sysdba sul container si crea l'utente e lo si attiva

`create user oved identified by password account unlock;`

`grant connect, resource to oved;`

`alter user oved quota unlimited on users;`

- Per terminare l'esecuzione di sqlplus
`exit`

TNSNAME.ORA

- in .../product/.../network/admin

ORCLPDB =

(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))

(CONNECT_DATA =

(SERVER = DEDICATED)

(SERVICE_NAME = orclpdb)

)

)

- refresh del listener Oracle

lsnrctl reload

- connessione via tnsname

sqlplus hr/hr@orclpdb

Alcuni IDE per database

- Quest Toad for Oracle
- Oracle SQL Developer
- DBeaver per Eclipse
- ...

Principali tipi di dati in Oracle

NUMBER(precision, scale)

INTEGER

CHAR(length)

VARCHAR2(length)

DATE

SELECT

`select sysdate from dual; -- la tabella dual`

`select * from regions; -- tutta la tabella`

- La selezione può essere filtrata per colonne e righe

`select region_name from regions; -- colonne`

`select * from regions where region_id = 1; -- righe`

- Pseudo colonne

`select rowid, rownum from regions;`

Dettagli su tabelle

- Tabelle dell'utente corrente

```
select table_name  
from user_tables;
```

- Descrizione di una tabella

```
describe jobs;
```

```
select column_name, nullable, data_type, data_length, data_precision, data_scale  
from user_tab_columns  
where table_name = 'JOBS';
```


Operatori aritmetici

- Numeri

```
select 1+2, 3-4, 2*6, 5/2  
from dual;
```

- Date

```
select to_date('30-AUG-2019') + 2, to_date('02-AUG-2019') - 3 from dual;  
select to_date('02-AUG-2018') - to_date('25-JUL-2018') from dual;
```

- Modifica i risultati in lettura da tabella

```
select job_title, min_salary, min_salary + 2000, min_salary * 3 + 1000  
from jobs;
```

Alias di colonna

- Introdotto da “as” (opzionale)

```
select job_title, min_salary as original, min_salary salary  
from jobs;
```

- Delimitato da doppi apici se include spazi e per mantenere il case (altrimenti maiuscolo)

```
select job_title, min_salary + 2000 "increased min salary"  
from jobs;
```

Concatenazione

- Operatore ||

```
select first_name || ' ' || last_name as "Employee's name"  
from employees;
```

NULL

- Valore non presente o non valido, check esplicito con “is null”

```
select first_name, last_name, commission_pct  
from employees  
where commission_pct is null;
```

- “Assorbe” altri operandi

```
select first_name, last_name, 12 * salary * commission_pct  
from employees;
```

- Il built-in NVL() evita ambiguità

```
select first_name, last_name, 12 * salary * nvl(commission_pct, 0)  
from employees;
```

SELECT DISTINCT

- Tutti i manager con dipendenti associati
`select manager_id`
`from employees;`
- DISTINCT ci permette di filtrare i duplicati
`select distinct manager_id`
`from employees;`

Operatori di confronto

- =, !=, <, >, <=, >=, ANY (almeno uno), ALL
select * from regions where region_id = 1;
select * from regions where region_id != 2;
select * from regions where region_id < 3;
select * from regions where region_id <= 3;
select * from regions where region_id > any(1, 2, 3);
select * from regions where region_id > all(1, 2, 3);

Operatori SQL

- LIKE, IN, BETWEEN, IS NULL. Per negare il loro risultato: NOT
- LIKE wildcard: _ %
select last_name from employees where last_name like '_ul%';
- IN
select * from regions where region_id not in (2, 3);
select * from regions where region_id not in (2, 3, null); -- !! NOT IN(..., NULL) → FALSE !!
- BETWEEN
select * from regions where region_id between 2 and 3;
- IS NULL
– select * from employees where manager_id is null;

Operatori logici

- AND

```
select * from employees  
where salary < 3000 and employee_id > 195;
```

- OR (disgiunzione inclusiva)

```
select * from employees  
where salary > 20000 or last_name = 'King';
```

- NOT

```
select * from employees  
where not department_id > 20;
```


Ordinamento via ORDER BY

- ORDER BY segue FROM – WHERE

```
select * from employees
```

```
order by last_name;
```

- ASC (ascending, default) / DESC (descinding)

```
select * from employees
```

```
order by last_name desc, first_name asc;
```

- notazione posizionale

```
select first_name, last_name from employees
```

```
order by 2;
```

JOIN

- Selezione di dati provenienti da diverse tabelle
- INNER JOIN – produce una riga se le colonne nella join contengono un valore che soddisfa la condizione (un NULL → riga scartata)
- OUTER JOIN – preservano righe con NULL
- SELF JOIN – left e right nella JOIN sono la stessa tabella
- NON-EQUI JOIN – usano operatori diversi da “=”

INNER JOIN

- Selezione dati correlati su diverse tabelle

```
select region_name from regions where region_id = 1;  
select country_name from countries where region_id = 1;  
-- region_id = 1 .. 4
```

- Equi-join “classica” sulla relazione PK → FK

```
select region_name, country_name  
from regions, countries  
where regions.region_id = countries.region_id;
```

Alias per tabelle

- Si possono definire nel FROM alias per tabelle validi solo per la query corrente

```
select r.region_name, c.country_name  
from regions r, countries c  
where r.region_id = c.region_id;
```

JOIN – USING vs NATURAL JOIN

- INNER JOIN standard SQL/92

```
select region_name, country_name  
from regions join countries -- join è “inner” per default  
using(region_id);
```

- Se la relazione è “naturale” → NATURAL JOIN

```
select region_name, country_name  
from regions natural join countries;
```

JOIN – ON

- NATURAL JOIN e JOIN – USING implicano una relazione equi-join per PK e FK con lo stesso nome
- JOIN – ON ci permette una maggior libertà

```
select region_name, country_name
```

```
from regions join countries
```

```
on(regions.region_id = countries.region_id);
```

JOIN – WHERE

- JOIN – ON

```
select region_name, country_name  
from regions r join countries c  
on(r.region_id = c.region_id)  
where r.region_id = 1;
```

- JOIN – USING

```
select region_name, country_name  
from regions join countries  
using(region_id)  
where region_id = 1;
```

- NATURAL JOIN

```
select region_name, country_name  
from regions natural join countries  
where region_id = 1;
```

- query classica equivalente

```
select region_name, country_name  
from regions r, countries c  
where r.region_id = c.region_id  
and r.region_id = 1;
```

Prodotto Cartesiano

- Se manca la condizione in una JOIN, ogni riga della prima tabella viene abbinata con tutte le righe della seconda

```
select region_name, country_name  
from regions, countries;
```

- SQL/92 CROSS JOIN, richiede che sia esplicito

```
select region_name, country_name  
from regions cross join countries;
```


Self JOIN

- La FK si riferisce alla PK della stessa tabella

```
select e.last_name as employee, m.last_name as manager  
from employees e join employees m  
on (e.manager_id = m.employee_id);
```

- Versione “classica”

```
select e.last_name as employee, m.last_name as manager  
from employees e, employees m  
where e.manager_id = m.employee_id;
```

JOIN su più tabelle

- JOIN – ON ha solo una tabella left e una right → 2 JOIN per 3 tabelle
select employee_id, city, department_name
from employees e join departments d on d.department_id = e.department_id
join locations l on d.location_id = l.location_id;
- Versione “classica” → 2 condizioni nel WHERE per 3 tabelle
select employee_id, city, department_name
from employees e, departments d, locations l
where d.department_id = e.department_id and d.location_id = l.location_id;

Non-equi JOIN

- JOIN basate su operatori diversi da “=”, poco usate
select e.last_name, e.salary, j.min_salary
from employees e join jobs j
on(e.job_id = j.job_id and e.salary between j.min_salary and j.min_salary + 100);
- Versione “classica”
select e.last_name, e.salary, j.min_salary
from employees e, jobs j
where e.job_id = j.job_id
and e.salary between j.min_salary and j.min_salary + 100;

LEFT OUTER JOIN

- Preserva i valori nella tabella left anche in caso di NULL nella tabella right

```
select first_name, department_name  
from employees left outer join departments  
using(department_id)  
where last_name = 'Grant';
```

RIGHT OUTER JOIN

- Preserva i valori nella tabella right anche in caso di NULL nella tabella left

```
select first_name, last_name, department_name  
from employees right outer join departments  
using(department_id)  
where department_id between 110 and 120;
```

FULL OUTER JOIN

- Preserva i valori di una tabella anche in caso di NULL nell'altra

```
select e.last_name, d.department_name  
from employees e full outer join departments d  
on (e.department_id = d.department_id)  
where last_name = 'Grant'  
or d.department_id between 110 and 120;
```

Funzioni su riga singola

- Operano su e ritornano una singola riga
 - Caratteri e stringhe
 - Numeri
 - Conversione
 - Date
 - Espressioni regolari

Alcune funzioni su stringhe

- **ASCII()**: codice ASCII di un carattere, **CHR()**: da codice ASCII a carattere
 - select ascii('A') as A, chr(90) as "90" from dual;
- **CONCAT()**: concatenazione di stringhe, cfr. operatore ||
 - select concat(first_name, last_name) from employees;
- **INITCAP()**: iniziali maiuscole, **UPPER()**: tutto maiuscolo, **LOWER()**: tutto minuscolo
 - select initcap('a new thing') as initcap, lower('NEW') low, upper('old') up from dual;
- **INSTR()**: x, target, start, occurrence → [1..n], 0 not found
 - select instr('crab', 'ba') as "not found", instr('crab abba rabid cab', 'ab', 2, 3) as pos from dual;
 - select instr(sysdate, '19') as pos from dual;
- **LENGTH()**: per string e numeri, convertiti implicitamente in stringhe
 - select length('name'), length(42000) from dual;

Alcune funzioni su stringhe /2

- **LPAD()**, **RPAD()**: padding. Stringa → dimensione, con eventuale pad
 - `select lpad('tom', 30, '.') tom, rpad('tim', 30, '_- _') tim from dual;`
- **LTRIM()**, **RTRIM()**, **TRIM()**: rimozione di caratteri dall'input
 - `select ltrim(' Hi!'), rtrim('Hi!abab', 'ab'), trim('0' from '00Hi!000') from dual;`
- **NVL()**: null to value, se null → secondo parametro
 - `select employee_id, nvl(commission_pct, 0) from employees;`
- **NVL2()**: se non è null → secondo parametro, altrimenti il terzo
 - `select employee_id, nvl2(commission_pct, 'value', 'no value') from employees;`
- **REPLACE()**: sostituzione di substring, **SUBSTR()**: estrazione di substring
 - `select replace('Begin here', 'Begin', 'End'), substr('ABCDEFGH',3,4) from dual;`

Alcune funzioni numeriche

- **ABS()**: valore assoluto
- **CEIL()**: 'soffitto', **FLOOR()**: 'pavimento'
- **MOD()**: modulo, resto di divisione intera
- **POWER()**: potenza, **EXP()**: e^x , **SQRT()**: radice quadrata, **LOG()**, **LN()**: logaritmi
- **ROUND()**, **TRUNC()**: arrotonda/tronca a n decimali o potenze di 10 se n è negativo
- **SIGN()**: -1, 0, 1 per numeri negativi, zero, positivi

```
select abs(10), abs(-10), ceil(5.8), ceil(-5.2), floor(5.8), floor(-5.2), mod(8, 3), mod(8, 4),  
       power(2, 1), power(2, 3), exp(1), sqrt(25), sqrt(5), log(10, 100), ln(exp(1)),  
       round(5.75), round(5.75, 1), round(5.75, -1), trunc(5.75), trunc(5.75, 1), trunc(5.75, -1)  
from dual;
```

Alcune funzioni di conversione

- **TO_CHAR()**, **TO_NUMBER()**: convertono (formattando) a VARCHAR2 e NUMBER
- **CAST()**, converte ad un tipo supportato da ORACLE, se compatibile

```
select to_char(12345.67), to_char(12345.67, '99,999.99'),  
       to_char(2019, 'RN'), to_number('970,13') * 2,  
       cast('05-JUL-18' as date) + 2, cast(12345.678 as number(10,2))  
from dual;
```

Alcune funzioni su date

- **ADD_MONTHS()**: aggiunge mesi alla data
- **MONTHS_BETWEEN()**: mesi tra le due date
- **NEXT_DAY()**: giorno della settimana successivo al corrente
- **LAST_DAY()**: ultimo giorno del mese
- **ROUND()**, **TRUNC()**: arrotonda/tronca il giorno

```
select add_months(sysdate, 3), months_between(sysdate, '01-FEB-2019'),  
       last_day(sysdate), next_day(sysdate, 'monday'),  
       round(sysdate, 'year'), round(sysdate, 'month'),  
       trunc(sysdate, 'year'), trunc(sysdate, 'month')  
from dual;
```

Espressioni regolari

- **REGEXP_LIKE()** versione estesa di LIKE
 - Es: cognomi che iniziano per A o E:
select last_name
from employees
where regexp_like(last_name, '^[AE].*');

Funzioni aggregate

- Ignorano i NULL
- Uso di DISTINCT per filtrare duplicati
- AVG(): media
- COUNT(): numero di righe
- MAX(): valore massimo
- MEDIAN(): mediana
- MIN(): minimo
- STDDEV(): deviazione standard
- SUM(): somma
- VARIANCE(): varianza

Raggruppamento via GROUP BY

- Divide il risultato della select in gruppi
- È possibile applicare funzioni aggregate sui gruppi
select department_id, trunc(avg(salary))
from employees
group by department_id
order by 1;

GROUP BY – HAVING

- HAVING filtra i risultati di GROUP BY
- È possibile filtrare prima le righe della SELECT con WHERE, e poi il risultato della GROUP BY con HAVING

```
select manager_id, trunc(avg(salary))
```

```
from employees
```

```
where salary < 8000
```

```
group by manager_id
```

```
having avg(salary) > 6000
```

```
order by 2 desc;
```


Subquery

- In WHERE:
select first_name, last_name from employees
where employee_id = (select manager_id from employees where last_name = 'Chen');
- In HAVING:
select department_id, trunc(avg(salary)) from employees
group by department_id having avg(salary) < (
select max(avg(salary)) from employees group by department_id);
- In FROM (inline view):
select employee_id
from (select employee_id from employees where employee_id between 112 and 115);

JOIN con subquery

- Subquery genera una tabella temporanea → join
select region_name, country_count
from regions natural join (
select region_id, count(rowid) country_count
from countries
group by region_id);

subquery multirighe in WHERE

- Uso degli operatori IN, ANY, ALL

es: nome di EMPLOYEES che sono manager

```
select first_name, last_name from employees
```

```
where employee_id in(
```

```
    select distinct manager_id
```

```
    from employees where manager_id is not null)
```

```
order by 2;
```

INSERT

`INSERT INTO table (columns...) VALUES (values...);`

`insert into regions(region_id, region_name)`

– `values (11, 'Antarctica');`

- I valori `NULLABLE`, se `NULL`, sono impliciti

`insert into regions(region_id) values (12);`

- Il nome delle colonne è opzionale (cfr. `DESCRIBE`)

`insert into regions values (13, null);`

UPDATE (WHERE!)

UPDATE table

SET column = value

[WHERE condition];

update regions

set region_name = 'Region ' || region_id

where region_id > 10;

DELETE (WHERE!)

```
DELETE FROM table [WHERE condition];
```

```
delete from regions  
where region_id > 10;
```

Transazioni

- Inizio: prima istruzione DML (INSERT, UPDATE, DELETE) in assoluto, o dopo la chiusura di una precedente transazione
- Fine: COMMIT, ROLLBACK, istruzione DDL, DCL, EXIT (implicano COMMIT o ROLLBACK in caso di failure)
- Buona norma: COMMIT o ROLLBACK esplicite

COMMIT, ROLLBACK, SAVEPOINT

SAVEPOINT: punto intermedio in una transazione

```
insert into regions(region_id, region_name) values (11, 'Antarctica');  
savepoint sp;
```

```
insert into regions(region_id, region_name) values (12, 'Oceania');
```

```
rollback to sp; -- keep Antarctica, rollback Oceania
```

```
commit; -- persist Antarctica
```


Livelli di isolamento nelle transazioni

- Transazioni concorrenti possono causare problemi in lettura:
 - Phantom read: T1 esegue una SELECT due volte, risultati diversi a causa di un INSERT di T2
 - Non repeatable: T1 select, T2 update, T1 select non ripetibile
 - Dirty: T1 update, T2 select, T1 rollback, valore per T2 è invalido
- Oracle supporta
 - READ COMMITTED**: no dirty read ← default Oracle
 - SERIALIZABLE: nessuno dei problemi indicati ← default SQL

CREATE TABLE

- Nome tabella, nome e tipo colonne, constraint, ...

```
create table simple (  
    simple_id integer primary key,  
    status char,  
    name varchar2(20),  
    coder_id integer);
```

CREATE TABLE AS SELECT

- Se si hanno i privilegi in lettura su una tabella si possono copiare dati e tipo di ogni colonna

```
create table coders
```

```
as
```

```
select employee_id as coder_id, first_name, last_name, hire_date, salary  
from hr.employees  
where department_id = 60;
```

ALTER TABLE

- ADD / DROP COLUMN

- `alter table simple add counter number(38, 0);`

- `alter table simple drop column counter;`

- ADD CONSTRAINT CHECK / UNIQUE

- `alter table simple add constraint simple_status_ck check(status in ('A', 'B', 'X'));`

- `alter table coders add constraint coders_name_uq unique(first_name, last_name);`

- MODIFY column CONSTRAINT NOT NULL

- `alter table simple modify name constraint simple_name_nn not null;`

- ADD CONSTRAINT PRIMARY KEY

- `alter table coders add constraint coders_pk primary key(coder_id);`

- DROP CONSTRAINT

- `alter table simple drop constraint simple_name_nn;`

ALTER TABLE per FK

- Creazione di una colonna come FK

```
alter table simple add constraint simple_coder_id_fk  
coder_id references coders(coder_id);
```

- FK con rimozione degli orfani

```
... coder_id references coders(coder_id) on delete cascade;
```

- FK con rimozione della relazione

```
... coder_id references coders(coder_id) on delete set null;
```

CREATE TABLE con CONSTRAINT

```
create table simple (  
  simple_id integer  
    constraint simple_pk primary key  
    constraint simple_id_ck check (mod(simple_id, 2) = 1),  
  status char default 'A'  
    constraint simple_status_ck check (status in ('A', 'B', 'X')),  
  name varchar2(20),  
    -- constraint simple_name_nn not null,  
    -- constraint simple_name_uq unique,  
  coder_id integer  
    constraint simple_coder_id_fk references coders(coder_id) on delete cascade,  
  
  constraint simple_name_status_uq unique(name, status));
```

TRUNCATE / DROP TABLE

- `TRUNCATE TABLE table_name; -- no rollback!`
- `DROP TABLE table_name; -- no rollback!`
- `DELETE FROM table_name; -- DML → rollback`

INDEX

- Velocizza accesso a tabella, consigliato per query che ritornano $< 10\%$ righe
- Andrebbero creati in un proprio tablespace, informazioni in USER_INDEXES
- **B-Tree**
 - consigliato per colonne con valori unici, usato da Oracle per PK

```
create index coders_last_name_ix on coders(last_name);  
create index coders_name_ix on coders(first_name, last_name);  
drop index coders_last_name_ix;
```
- **Bitmap**
 - più efficienti per colonne con pochi valori

```
create bitmap index coders_gender_ix on coders(gender);
```


SEQUENCE

- Oggetto di database che genera una sequenza di interi
`create sequence my_seq; -- inizia da 1, incremento 1`
- `nextval`: incrementa e ritorna il valore della sequenza
`select my_seq.nextval from dual;`
- `currval`: ritorna il valore corrente, senza incremento
`select my_seq.currval from dual;`
- Le sequenze possono essere modificate o eliminate
`alter sequence my_seq increment by 2;`
`drop sequence my_seq;`

SEQUENCE /2

- Sequenza con custom start e increase:
`create sequence my_seq start with 201 increment by 2;`
- Altre proprietà definibili su di una sequenza:
minvalue, maxvalue, cycle, order, etc.
- **PK**: si delega alla sequenza la generazione di valori univoci
insert into coders
values(my_seq.nextval, 'Bertrand', 'Meyer', SYSDATE, 8000);
- Info nella tabella USER_SEQUENCES

VIEW

- Query predefinita su una o più tabelle, acceduta come se fosse una tabella
- Semplifica e controlla l'accesso ai dati

```
create or replace view odd_coders_view as
```

```
select * from coders
```

```
where mod(coder_id, 2) = 1
```

```
with read only;
```

```
drop view odd_coders_view;
```

PL/SQL

- Estensione procedurale di Oracle a SQL
- Basato sulla definizione di blocco

[DECLARE

...]

variabili locali

BEGIN

...

Loop, logica condizionale.
Ogni istruzione è terminata
da un punto e virgola

[EXCEPTION

gestione degli errori

...]

END;

terminatore di blocco

/

Hello PL/SQL

- Un blocco minimale
 - non sono necessarie DECLARE e EXCEPTION
- By default l'output su console non è attivo

```
set serveroutput on

begin
    dbms_output.put_line('Hello PL/SQL');
end;
/
```

Variabili

- Le variabili sono dichiarate nel blocco DECLARE
- Tutti i tipi SQL di Oracle sono supportati da PL/SQL, più alcuni tipi aggiuntivi, come BOOLEAN
- La loro definizione non è obbligatoria
- Per convenzione iniziano per “v_”

```
declare
    v_width integer;
    v_height integer := 2;
    v_area integer := 6;
begin
    v_width := v_area / v_height;
    dbms_output.put_line(
        'v_width = ' || v_width);
end;
/
```

Eccezioni

- Gestione degli errori di esecuzione
- Simile al meccanismo try/catch di Java

```
begin
    dbms_output.put_line(6 / 0);
exception
    when zero_divide then
        dbms_output.put_line('Zero divide!');
end;
/
```

```
begin
    dbms_output.put_line(1 / 0);
exception
    when others then
        dbms_output.put_line('Exception!');
end;
/
```

IF – ELIF – ELSE – END IF

```
declare
  v_a integer := 1;
begin
  if v_a > 0 then
    dbms_output.put_line('v_a is positive');
  elsif v_a = 0 then
    dbms_output.put_line('v_a is zero');
  else
    dbms_output.put_line('v_a is negative');
  end if;
end;
/
```


LOOP

- Loop semplice: EXIT (WHEN), CONTINUE (WHEN)

```
v_x := 0;  
loop  
  v_x := v_x + 1;  
  if v_x = 3 then exit;  
end if;  
end loop;
```

```
v_x := 0;  
loop  
  v_x := v_x + 1;  
  exit when v_x = 5;  
end loop;
```

```
v_x := 0;  
loop  
  v_x := v_x + 1;  
  if v_x = 3 then  
    -- something special  
    continue;  
  end if;  
  exit when v_x = 5;  
end loop;
```

```
v_x := 0;  
loop  
  v_x := v_x + 1;  
  continue when v_x = 3;  
  
  -- something normal  
  exit when v_x = 5;  
end loop;
```

WHILE e FOR

- WHILE LOOP, finché la condizione è vera
- FOR LOOP, per ogni valore indicato (REVERSE)

```
v_x := 0;  
while v_x < 5 loop  
    v_x := v_x + 1;  
end loop;
```

```
for i in 1..5 loop  
    dbms_output.put_line('for loop: ' || i);  
end loop;
```

```
for i in reverse 1..5 loop  
    dbms_output.put_line('for loop: ' || i);  
end loop;
```

SELECT INTO

- Lettura di una singola riga

```
declare
  v_first_name coders.first_name%type;
  v_last_name coders.last_name%type;
begin
  select first_name, last_name
  into v_first_name, v_last_name
  from coders
  where coder_id = 103;

  dbms_output.put_line([' | v_first_name | ' | v_last_name | ']);
end;
/
```

Tipo di una colonna

CURSOR

- Lettura di più righe
- Si definisce un CURSOR associato a SELECT
- OPEN CURSOR esegue la SELECT
- FETCH – INTO legge la riga corrente
- EXIT WHEN %NOTFOUND termina la lettura del cursore
- CLOSE CURSOR rilascia le risorse associate

```
declare
    v_last_name coders.last_name%type;
    v_hire_date coders.hire_date%type;
    cursor v_coder_cursor is
        select last_name, hire_date from coders;
begin
    open v_coder_cursor;
    loop
        fetch v_coder_cursor
        into v_last_name, v_hire_date;
        exit when v_coder_cursor%notfound;

        dbms_output.put_line(
            '[' || v_last_name || ', ' || v_hire_date || ']');
    end loop;
    close v_coder_cursor;
end;
/
```

CURSOR in FOR LOOP

- gestione implicita, codifica semplificata

OPEN cursor
implicita

```
declare
    cursor v_coder_cursor is
        select last_name, hire_date from coders;
begin
    for v_cur in v_coder_cursor loop
        dbms_output.put_line(
            '[' || v_cur.last_name || ', ' || v_cur.hire_date || ']');
    end loop;
end;
/
```

CLOSE cursor
implicita

CREATE PROCEDURE

Parametri IN / OUT

PROCEDURE
body

```
create or replace procedure get_coder_salary(  
  p_coder_id in coders.coder_id%type,  
  p_salary out coders.salary%type) is  
begin  
  select salary  
  into p_salary  
  from coders  
  where coder_id = p_coder_id;  
end get_coder_salary;  
/
```

IS / AS

```
drop procedure get_coder_salary;
```

Esecuzione di una procedura

```
declare
  v_id coders.coder_id%type := 105;
  v_salary coders.salary%type;
begin
  get_coder_salary(v_id, v_salary);
  dbms_output.put_line('Salary is ' || v_salary);
exception
  when others then
    dbms_output.put_line('Can''t get salary for ' || v_id);
end;
/
```

CREATE FUNCTION

Parametri IN / OUT

Return type

FUNCTION
body

```
create or replace function get_salary(  
  p_coder_id in coders.coder_id%type)  
  return number as  
  v_salary coders.salary%type;  
begin  
  select salary  
  into v_salary from coders  
  where coder_id = p_coder_id;  
  return v_salary;  
end get_salary;  
/
```

IS / AS

Variabili locali

```
drop function get_salary;
```


Esecuzione di una funzione

```
declare
  v_id coders.coder_id%type := 105;
  v_salary coders.salary%type;
begin
  v_salary := get_salary(v_id);
  dbms_output.put_line('Salary is ' || v_salary);
exception
  when others then
    dbms_output.put_line('Can''t get salary for ' || v_id);
end;
/
```

TRIGGER

- Procedura eseguita automaticamente in relazione (prima, dopo, o invece) all'esecuzione di un comando DML
- Row-level
 - Eseguito per ogni riga coinvolta
 - In update, accesso a stato precedente e successivo
 - Esecuzione condizionale
- Statement-level
 - Eseguito una volta per tutte le righe

Un esempio di trigger

Tabella di output del trigger

```
create table coder_salaries (  
  coder_id number(6, 0)  
  references coders(coder_id),  
  old_salary number(8, 2),  
  new_salary number(8, 2)  
);
```

Trigger

```
create or replace trigger salary_update  
before update of salary on coders  
for each row  
begin  
  insert into coder_salaries values(  
    :old.coder_id, :old.salary, :new.salary);  
end salary_update;  
/
```

Generazione di eventi che scatenano il trigger

```
update coders  
set salary = salary * 1.3  
where coder_id > 103;
```