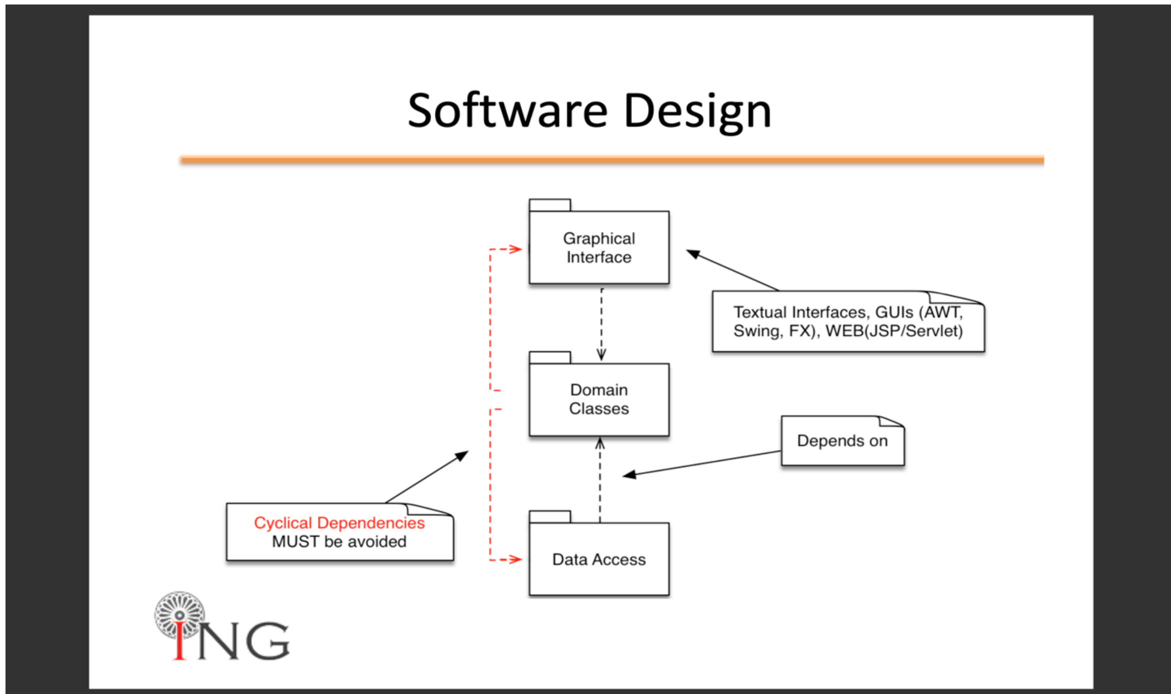
MODULO 7

1. STRUTTURA E APPLICAZIONI



- **Interfaccia grafica:** fornisce all'utente accesso alle classi del dominio → dipendono da classi del dominio
- **Accesso ai dati:** accesso ai dati, cioè classi che forniscono mapping DB e classi di dominio → dipendono dalle classi del dominio
- **Classi di dominio:** classi del dominio del problema → devono essere auto-contenute e riferirsi solo a loro stesse

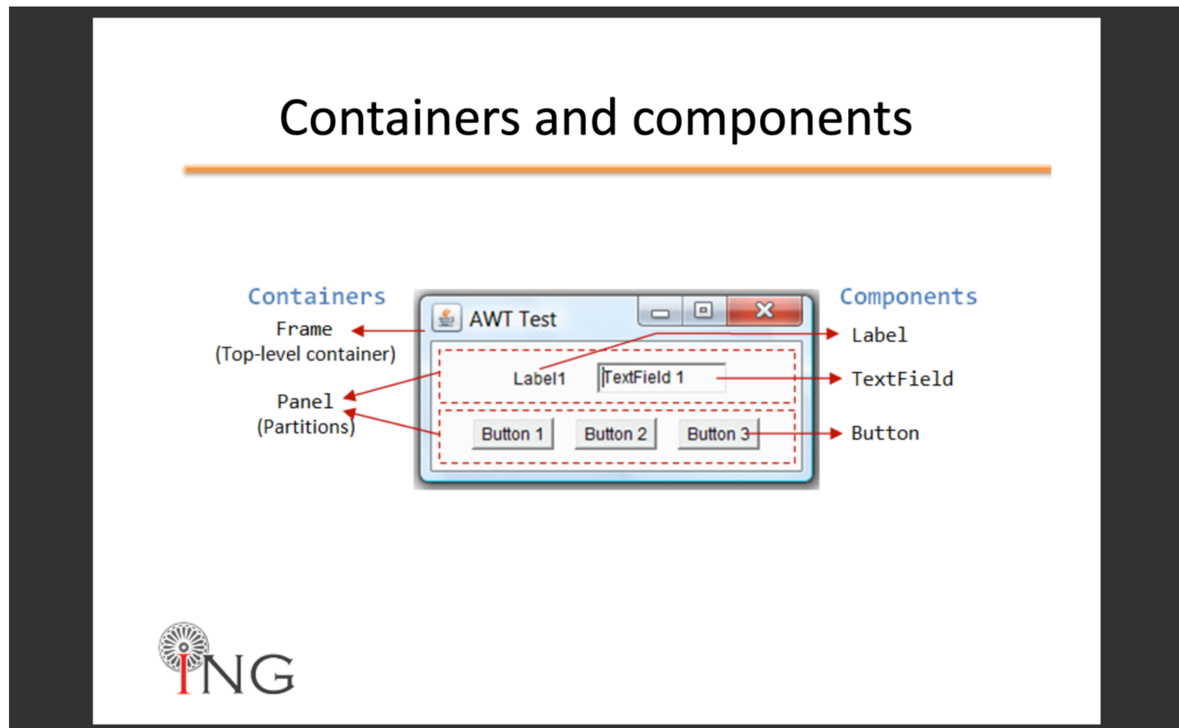
2. COMPONENT E CONTAINER

Package java.awt.* → offre le componenti per fare interfaccia grafica:

- **Componenti** (pulsante, casella di controllo, barra di scorrimento..)
- **Contenitori** (sono ancora componenti)
- **Gestione di eventi:**
  a) *Eventi generati dal sistema*
  b) *Eventi generati dall'interfaccia dell'utente*
- **Gestione del layout**

Package javax.swing.* → contiene gli *stessi componenti di java.awt.*,* ma con *nomi differenti* (JButton, JFrame..), *derivanti da* ***JComponent.*** → **è un'estensione di AWT**

- **Vantaggi:**
  a) Fornisce una serie di componenti con lo stesso aspetto e comportamento su tutte le piattaforme
  b) Aspetto modificabile in fase di esecuzione



→ come fare un'interfaccia grafica?

Processo iterativo:

- *Settare Look & Field = stile*
- *Definire uno o più top-level container → JFrame*
- *Aggiungo componenti semplici oppure compongo interfaccia costruendo container secondari*

3. SWING HELLO WORLD

Metodi base JFrame:

1. setcontentpane (Container c): setta ContentPane. È un contenitore secondario
2. add(Component c): aggiunge un componete a ContentPane
3. setDefaultCloseOperation (WindowConstants)
- EXIT_ON_CLOSE
- DO_NOTHING_ON_CLOSE

- DISPOSE_ON_CLOSE
- HIDE_ON_CLOSE

4. setSize (int base, int height): definisce dimensioni definitive del componente
5. setVisible (boolean visibility): definisce lo stato di visibilità del componente

4. BARRA DI MENU'

5. DIALOGHI

Le finestre di dialogo sono una scelta migliore rispetto all'istanza di altri JFrame!

- Ogni finestra di dialogo dipende da un contenitore di livello superiore.
- Le finestre di dialogo sono tutte istanze di JDialog, anche se la maggior parte viene eseguita utilizzando le classi di supporto (ad es. JOptionPane).

→ come realizzare finestre di dialogo?

- ***Specializzare JDialog*** *(contenitore di livello superiore) e definire i propri layout. Stesso principio della specializzazione di JFrame*
- ***Utilizzo di JOprionPane****, che fornisce supporto per la creazione di finestre di dialogo standard, fornendo icone, specificando titolo e testo della finestra di dialogo e personalizzando il testo del pulsante.*

6. LAYOUT

Le GUI predefinite, quando ridimensionate, consentono il trasferimento automatico dei componenti:

- questa è una necessità: Java funziona su molte piattaforme diverse. Android, ad esempio, funziona su TV da 65 "o smartphone da 6" (dimensioni dello schermo molto diverse!)

- *Inoltre, è possibile che il responsabile abbia individuato i componenti in un contenitore:* Flow, Border, Grid, GridBag, Layout delle carte
- JPanels sono contenitori che supportano layout: Diversi pannelli possono avere diversi gestori di layout - I gestori di layout vengono passati al costruttore JPanel.

7. MODELLO AD EVENTI

- Gli eventi sono ***classificati per tipo:*** MouseEvent, KeyEvent, ActionEvent
- Gli eventi vengono ***generati nei componenti*** (fonte)
- I ***Listeners*** (target) possono essere registrati su componenti
- ***Ogni volta che si verifica un evento, il thread dell'evento invia un messaggio a tutti i listeners registrati*** (l'evento viene passato come parametro)
- ***Gli ascoltatori devono implementare interfacce*** appropriate per rendere possibile il meccanismo di richiamata

→ come gestire gli eventi in Java?

Il principio che sottostà agli eventi è abbastanza simile alle eccezioni:

- i destinatari dichiarano quale evento è in grado di affrontare (una o più) implementando le interfacce necessarie
- i componenti che sono fonte di eventi (JButton, JTextField, ecc.) Selezionano i loro ricevitori → button.addActionListener (ricevitore)

Fai attenzione! Stai implementando interfacce, quindi devi implementare tutti i metodi di quelle interfacce!

8. MODELLI DI DELEGA EVENTI

→ I componenti possono:

1. **Gestire gli eventi da soli**: in caso di un gran numero di componenti
2. **Delegare eventi nel loro contenitore:** in caso di piccolo/medio numero di componenti
3. **Delegare eventi a classi esterne:** usato raramente perché produce classi non necessarie

MODULO 8

1. JDBC ARCHITETTURA

La maggior parte dei DBMS utilizza il protocollo di trasporto TCP. Accettano connessioni in entrata su una porta TCP specifica. Ciò consente connessioni sia locali che remote.

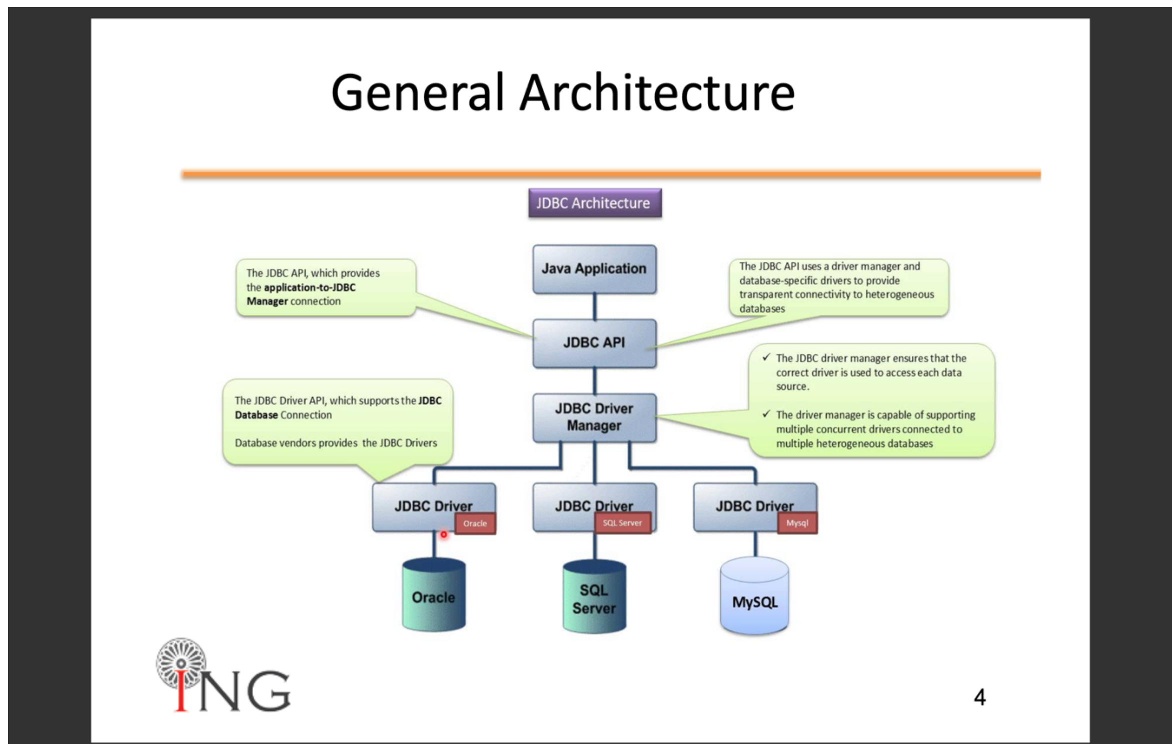
→ cos'è JDBC?

"Un'API che ti consente di accedere praticamente a qualsiasi origine di dati tabulari dal linguaggio di programmazione Java"

→ **Che cos'è un'API?** Interfaccia di programmazione dell'applicazione

→ **Che cos'è un'origine dati tabulare?** Database relazionali, fogli di calcolo, file flat

ARCHITETTURA GENERALE:



I driver JDBC forniscono la connessione al database e implementano il protocollo per il trasferimento di query e risultati tra il client e il database.

- Esistono 4 tipi di driver. Ci riferiamo a Tipo 4: Pure Java (vedi Appendice II)
- Ogni database ha bisogno di un driver specifico. Devono essere scaricati separatamente
- I driver sono librerie binarie Java e devono essere inclusi in CLASSPATH

2. UTILIZZO DI DRIVER O LIBRERIE ESTERNE

3. JDBC CONNECTION, STATEMENT E RESULTSET

Step base:

1. Caricare il driver specifico del fornitore

import java.sql.*;

/* this is for MySQL*/ Class.forName(“com.mysql.jdbc.Driver”);

/* this is for SQLite */ Class.forName(“org.sqlite.jdbc”);

→ JDBC è un'API astratta composta principalmente da interfacce e classi astratte. Le implementazioni concrete sono fornite principalmente all'interno dei driver.

2. Stabilire la connessione

DriverManager.getConnection(String url);

/* this is for MySQL*/
Connection c = DriverManager.getConnection(
“jdbc:mysql://localhost/**dbname**?user=**user**&password=**pass**”);

/* this is for SQLite */
Connection c = DriverManager.getConnection(“jdbc:sqlite:filename.db”);

→ Stabilisce una connessione a un database mediato dall'interfaccia Connessione. Il driver implementa l'interfaccia di connessione definita in JDBC

3. Crea una dichiarazione

Statement statement = c.createStatement() ;

→ Le interfacce JDBC Statement, CallableStatement e PreparedStatement definiscono i metodi e le proprietà che consentono agli sviluppatori di inviare comandi SQL o PL / SQL e ricevere dati dal database.
Definiscono inoltre metodi che aiutano a colmare le differenze tra i tipi di dati tra i tipi di dati Java e SQL utilizzati in un database.

4. Istruzioni ExecuteSQL

Dopo aver creato un oggetto Statement, è possibile utilizzarlo per eseguire un'istruzione SQL con uno dei suoi metodi:

- **int executeUpdate (String SQL):** *utilizzato per scrivere il database.* Utilizzare questo metodo per eseguire istruzioni SQL come INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE, ecc. Restituisce il numero di righe interessate dall'esecuzione dell'istruzione SQL.
- **ResultSet executeQuery (String SQL):** *utilizzato per la lettura del database.* Utilizzare questo metodo per eseguire istruzioni SQL come SELECT. Restituisce un oggetto ResultSet.

5. Get Result Set

L'interfaccia java.sql.ResultSet rappresenta il set di risultati di una query del database. Gli oggetti che implementano l'interfaccia ResultSet mantengono un cursore che punta alla riga corrente nel set di risultati.

- Metodi di navigazione: utilizzati per spostare il cursore attorno al ResultSet → *scelgo la riga*

- Ottieni metodi (Get methods): utilizzato per visualizzare i dati nelle colonne della riga corrente puntati dal cursore →*prelevo i dati dalla colonna della riga scelta*

OSS: resultSet.getXXX (), doveXXX è un tipo di dati primitivo. Le *colonne* possono essere *selezionate* tramite *nome o ID.*

- Metodi di aggiornamento: *utilizzati per aggiornare i dati nelle colonne della riga corrente*. Gli aggiornamenti sono scritti in modo trasparente all'interno del database sottostante (se supportato)

6. Chiusa la connessione

statement.close(); connection.close();

I programmi dovrebbero recuperare dagli errori e lasciare sempre il database in uno stato coerente. Gli errori di runtime devono essere ridotti al minimo nelle applicazioni industriali!

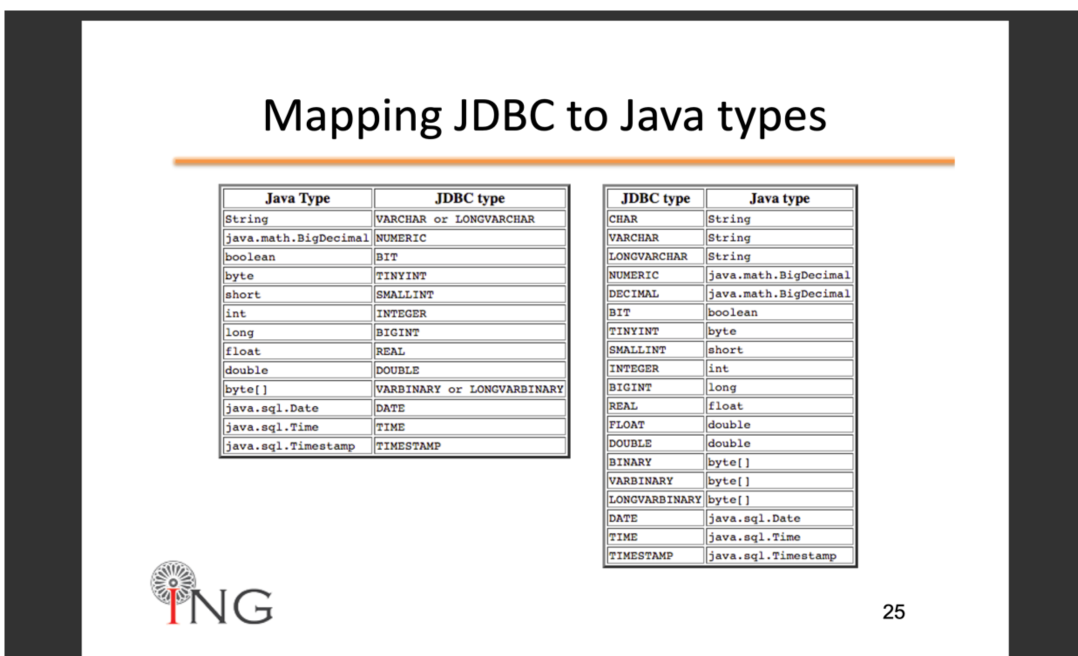
- Se un'istruzione genera un'eccezione, deve essere catturata in un'istruzione catch.
- La **clausola finally {...}** può essere utilizzata per lasciare il database in uno stato coerente.

I TIPI:

Esistono variazioni significative tra i tipi SQL supportati da diversi prodotti di database.

Ad esempio, la maggior parte dei database principali supporta un tipo di dati SQL per valori binari di grandi dimensioni, ma Oracle chiama questo tipo LONG RAW, Sybase lo chiama IMAGE e Informix lo chiama BYTE.

- I programmatori JDBC programmano principalmente con tabelle di database esistenti e non devono preoccuparsi degli esatti nomi di tipi SQL utilizzati.
- ***L'unico posto in cui i programmatori potrebbero aver bisogno di usare nomi di tipo SQL è nell'istruzione SQL CREATE TABLE quando stanno creando una nuova tabella di database.*** In questo caso i programmatori devono aver cura di usare nomi di tipo SQL supportati dal loro database di destinazione.



Java Type	JDBC type
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

JDBC type	Java type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

6. JDBC ResultSet AVANZATI

ResultSet sono oggetti simili a iteratori
- Non è possibile tornare indietro e andare avanti con il predefinito (TYPE_FORWARD_ONLY) ResultSet → È possibile chiamare solo next ()

- Non è possibile modificare i dati e in modo trasparente, il database → I dati devono essere manipolati in memoria e archiviati con un'altra operazione (statement.executeUpdate ())

1. DatabaseMetaData object

Un oggetto Connection fornisce un oggetto DatabaseMetaData in grado di fornire informazioni sullo schema che descrivono:

- Tavoli
- Grammatica SQL supportata
- Capacità supportate della connessione
- Procedure memorizzate

2. ResultSetMetaData

Un oggetto ResultSet fornisce un ResultSetMetaData, oggetto che fornisce informazioni sullo schema:

- Utile per scrivere codice in esecuzione su diverse tabelle.

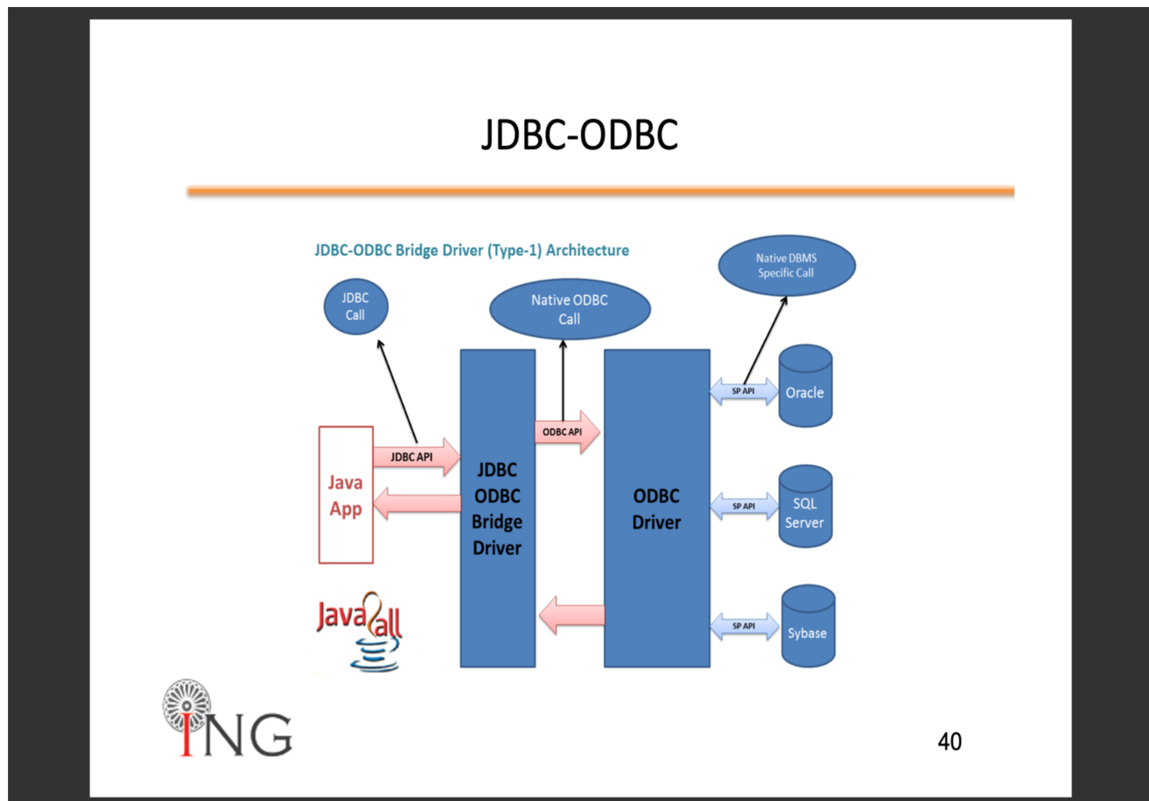
TRANSAZIONI: *Una transazione è un insieme di azioni da eseguire come singola azione atomica. O tutte le azioni vengono eseguite o nessuna di esse lo è.*

TRANSAZIONI JDBC:
JDBC consente di raggruppare le istruzioni SQL in un'unica transazione
- Il controllo delle transazioni viene eseguito dall'oggetto Connection, la modalità predefinita è il commit automatico, ovvero ogni istruzione sql viene trattata come una transazione
- È possibile disattivare la modalità di commit automatico con ***connection.setAutoCommit (false);***
- E riaccenderlo con ***connection.setAutoCommit (true);***
- ***Una volta che il commit automatico è disattivato, nessuna istruzione SQL verrà impegnata fino a quando un esplicito è invocato connection.commit ().*** A questo punto tutte le modifiche apportate dalle istruzioni SQL verranno rese permanenti nel database.

ARCHITETTURA GENERALE:

Cosa succede se devo utilizzare un DBMS raro che non è supportato da JDBC? (ad es. nessun driver rilasciato)? Usa **ODBC!**

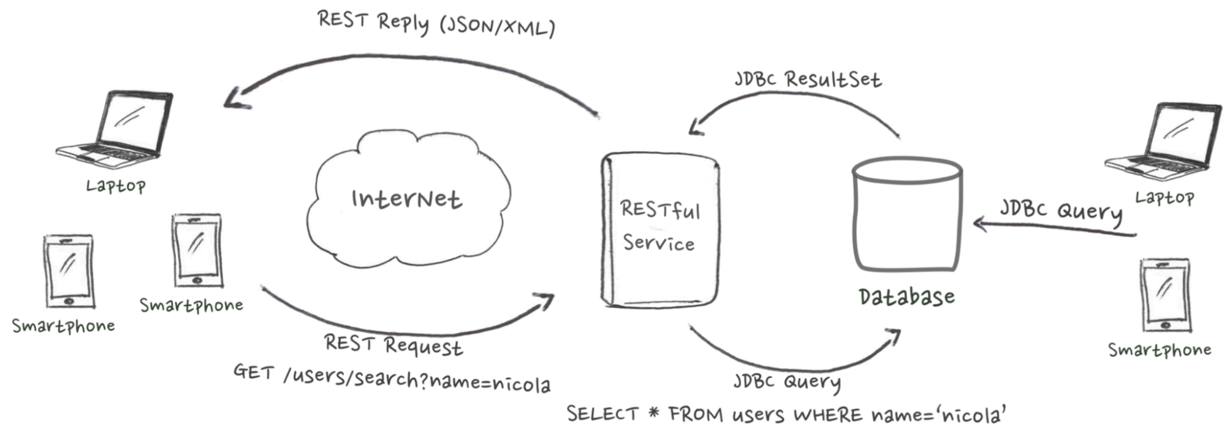
- Open Database Connectivity (ODBC) è un'API (Application Programming Interface) standard per l'accesso ai sistemi di gestione del database (DBMS).



→ ***Un bridge JDBC-ODBC è costituito da un driver JDBC che utilizza un driver ODBC per connettersi a un database di destinazione.*** Questo driver traduce le chiamate del metodo JDBC in chiamate di funzione ODBC. I programmatori di solito usano un tale bridge quando un dato database manca di un driver JDBC, ma è accessibile tramite un driver ODBC

MODULO 9

1. REST - CONCETTI GENERALI



- **REST viene utilizzato per creare servizi Web leggeri e scalabili**
- REST disaccoppia le applicazioni dai dettagli specifici del fornitore (ad es. JDBC) e impedisce di esporre DMBS a reti non attendibili (ad es. Internet)
- Librerie ampiamente disponibili per molte lingue (ad es. RESTLet per Java)

2. REST – RISORSE, VERBI, RAPPRESENTAZIONI

Concetti principali:

1) **Messaggi:** client e service comunicano attraverso scambio di messaggi basato su HTTP (richiesta e risposta).
   → I client inviano una richiesta al server e il server risponde con una risposta. Oltre ai dati effettivi, questi messaggi contengono anche alcuni metadati relativi al messaggio.

2) **Risorse:** vengono usate da ogni sistema e sono indirizzi a cui accediamo, che coincidono coi vari tipi di dato.
   → Le risorse possono essere immagini, video, dati degli utenti ecc ... Lo scopo di un servizio è quello di fornire un accesso alle risorse ai propri clienti. Gli architetti e gli sviluppatori di servizi desiderano che i servizi siano facili da implementare, gestibili, estensibili e scalabili.

   http://servicename/apiversion/resourse/id|service

3) **Rappresentazioni:** come viene rappresentato il dato che si ritorna
   → L'obiettivo di un servizio RESTful è sulle risorse e su come fornire accesso a tali risorse. Una risorsa può essere pensata come un oggetto come in OOP. Una risorsa può essere costituita da altre risorse.

Procedimento:

- Durante la progettazione di un sistema, *la prima cosa da fare è identificare le risorse e determinare in che modo sono correlate tra loro.*

(È simile alla progettazione di un database: identificare entità e relazioni.)

- Una volta identificate le nostre risorse, *la prossima cosa di cui abbiamo bisogno è trovare un modo per rappresentare queste risorse.*

È possibile utilizzare qualsiasi formato per rappresentare le risorse, in quanto REST non pone restrizioni al formato di una rappresentazione. Tuttavia, le ***rappresentazioni più utilizzate sono XML e JSON***

4) **Operazioni: I verbi HTTP (vedi Richiesta HTTP) definiscono le operazioni su risorse specifiche.**

   Metodi/verbi http:

1) **SAFE:** non effettuano nessun tipo di cambiamento della risorsa sul server (GET)
2) **INDEMPOTENTI:** ha stesso effetto indipendentemente dal numero di volte che viene effettuata la chiamata (DELETE E PUT).

Questa classificazione di metodi, semplifica la predizione dei risultati nell'ambiente inaffidabile del Web in cui il client può eseguire nuovamente la stessa richiesta.

OSS: METODO PUT E POST:

Non esiste alcuna differenza tra PUT e POST se la risorsa esiste già, entrambe aggiornano la risorsa esistente.

5) **Tecnologia senza stato: Un servizio RESTful è senza stato e non mantiene lo stato dell'applicazione per nessun client.**
   → Ogni richiesta è indipendente dalle altre.

3. REST – BUONE PRATICHE

Indirizzamento delle risorse (URI):

REST richiede che ogni risorsa abbia almeno un URI, il cui compito è identificare una risorsa o una raccolta di risorse.

Un servizio RESTful utilizza una gerarchia di directory come URI leggibili per indirizzare le proprie risorse. **L'operazione effettiva è determinata da un verbo HTTP. L'URI non dovrebbe dire nulla sull'operazione o sull'azione**.

→ Supponiamo di avere un database di persone e desideriamo esporlo al mondo esterno attraverso un servizio. **http://MyService/Persons/1**

Protocollo: http://ServiceName/ResourceType/ResourceID

Nomi x risorse:

- Usa ***nomi plurali*** per nominare le tue risorse.
- ***Evitare di usare gli spazi*** poiché creano confusione. Usa un _ (trattino basso) o - (trattino) invece.
- Un URI ***non distingue tra maiuscole e minuscole***; a cui risponde con stessa pagina.
- ***Un URI interessante non cambia mai***; quindi rifletti prima di decidere gli URI per il tuo servizio. Se è necessario modificare il percorso di una risorsa, non eliminare il vecchio URI e reindirizzare il client al nuovo percorso.
- ***Evitare i verbi per i nomi*** delle risorse fino a quando la risorsa non è effettivamente un'operazione o un processo. I verbi sono più adatti per i nomi delle operazioni.

Query parameters:
Lo scopo di base dei parametri della query è ***fornire parametri a un'operazione che necessita degli elementi di dati.***
- http://MyService/Persons/1?Format=json

- http://MyService/Persons/search?Name="nicola"

→ L'inclusione della codifica nell'URI principale non è logicamente corretta:
- http: // MyService / Persone / 1 / json /

Documentazione:
Non ci sono scuse per non documentare il tuo servizio.
→ È necessario documentare ogni risorsa e URI per gli sviluppatori client. È possibile utilizzare qualsiasi formato per strutturare il documento, ma dovrebbe contenere informazioni sufficienti su risorse, URI, metodi disponibili e qualsiasi altra informazione richiesta per accedere al servizio.

Vantaggi:

REST è un ottimo modo per sviluppare servizi Web leggeri che sono facili da implementare, mantenere e scoprire.

HTTP fornisce un'interfaccia eccellente per implementare servizi RESTful con funzionalità come un'interfaccia uniforme e memorizzazione nella cache. Tuttavia, spetta agli sviluppatori implementare e utilizzare queste funzionalità correttamente.

→ Se le basi sono giuste, un servizio RESTful può essere facilmente implementato utilizzando una qualsiasi delle tecnologie esistenti come Python, .NET o Java.

Svantaggi:

1) Nessun supporto per le transazioni
- Supporto DBMS (generalmente dietro i servizi REST) transazioni

2) Nessun supporto per la pubblicazione / sottoscrizione.
- La notifica viene effettuata tramite polling.
- Il client può eseguire il polling del server. GET è estremamente ottimizzato sul web.

3) Elevata larghezza di banda
- HTTP utilizza un modello di richiesta / risposta, quindi ci sono molti bagagli che volano intorno alla rete per far funzionare tutto.

MODULO 10

1. I/O FILE, STREAM, FRAMEWORK ASTRATTO

a) **Astrazione ad accesso casuale**
   - java.io.RandomAccessFile

b) **Astrazione orientata al flusso**
   - Le operazioni di I/O funzionano allo stesso modo con tutti i tipi di flussi. Ad esempio, un flusso può essere: input standard, output, errore, file, dati da / verso la memoria o una pipe, una connessione di rete

1) Classi astratte: Reader Writer
   - Per leggere e scrivere flussi di caratteri (Unicode carattere a 16 bit)

2) Classi astratte: InputStream OutputStream
   - Per leggere e scrivere flussi di byte (8 bit)

Package java.io
Le eccezioni sono sottoclassi di java.io.IOException

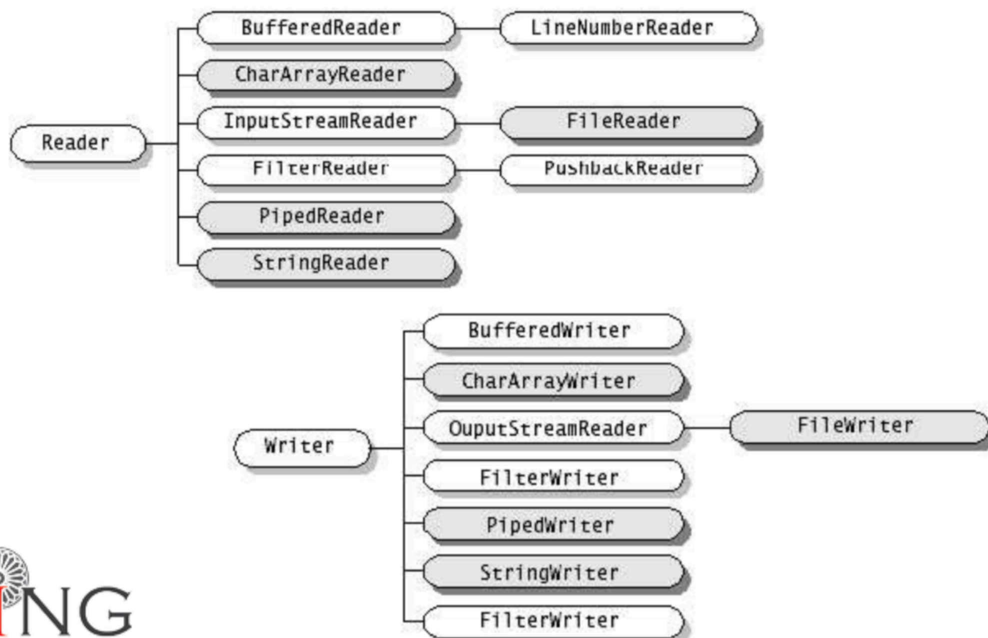
java.io.PrintStream: aggiunge funzionalità a un flusso di output generico. In particolare la capacità di stampare rappresentazioni convenienti di vari dati.

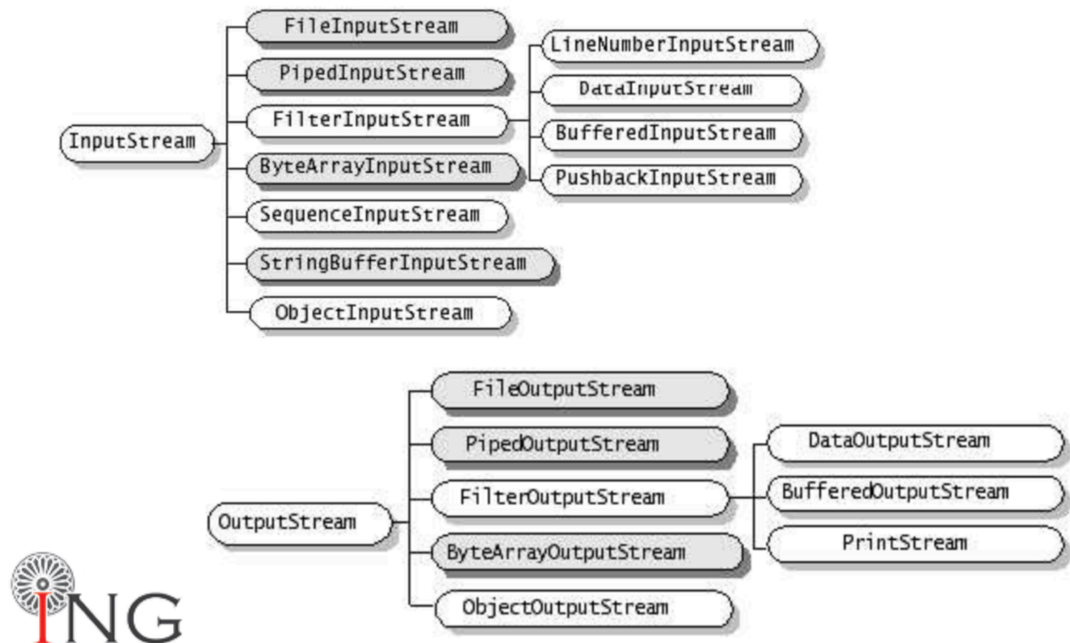
Aggiunge metodi come print (), println (), printf (), format () per la formattazione delle stringhe.

A differenza di altri flussi di output, PrintStream non genera mai un'eccezione IOException; invece, situazioni eccezionali semplicemente impostano un flag interno che può essere testato tramite il metodo checkError.

2. I/O SORGENTI E DESTINAZIONI DEI DATI

Reader and Writer





3. I/O BUFFERED STREAMS

I flussi bufferizzati aggiungono funzionalità di buffering. L'alternativa manuale è usare read (int [] buf) o read (char [] buf).

- BufferedInputStream
- BufferedOutputStream
- BufferedReader
- BufferedWriter

4. I/O INTERPRETATED STREAMS

Traduce i tipi di limiti predefiniti (UTF-8) su un flusso

- **DataInputStream (InputStream i):**
  readByte (), readChar (), readDouble (), readFloat (), readInt (), readLong (), readShort ()

- **DataOutputStream (OutputStream o):**
  writeByte (), writeChar (), writeDouble (), writeFloat (), writeInt (), writeLong (), writeShort ()

5. I/O OBJECT STREAMS

Serializzazione:
Lettura/scrittura di un oggetto implica:
- attributi di lettura / scrittura (e facoltativamente il tipo) di oggetto
- Separare correttamente diversi elementi
- Durante la lettura, crea un oggetto e imposta tutti i valori degli attributi

→ **Queste operazioni (serializzazione) sono automatizzate da:**
ObjectInputStream
ObjectOutputStream

I metodi per leggere/scrivere oggetti sono:
- void writeObject (Object)
- Oggetto readObject ()

È possibile serializzare SOLO oggetti che implementano l'interfaccia Serializable. Serializable è un'interfaccia vuota. Viene utilizzato per evitare la serializzazione di oggetti, senza l'autorizzazione dello sviluppatore della classe

Un ObjectOutputStream salva automaticamente tutti gli oggetti indicati dai suoi attributi
- gli oggetti serializzati sono numerati nel flusso
- i riferimenti vengono salvati come numeri d'ordine nel ruscello

OSS: Se salvo 2 oggetti che indicano un terzo, questo viene salvato solo una volta:
- Prima di salvare un oggetto, ObjectOutputStream verifica se non è già stato salvato
- Altrimenti salva solo il riferimento (come numero)

6. I/O UTILITA'

Una rappresentazione astratta di nomi di file e directory. Fornisce l'accesso ad alcuni attributi di file (lunghezza, diritti, ecc.) E una mappatura tra File e String

- Platform dependent

– File f = new File("tmp/abc.txt");

• Platform independent
– File f = new File(“tmp” + File.separator + "abc.txt");

Java.nio.file.Files:
Questa classe consiste esclusivamente di metodi statici per manipolare file o directory (copia, sposta, elimina, rinomina, imposta la proprietà, imposta gli attributi)
→ Nella maggior parte dei casi, i metodi qui definiti delegano al provider del file system associato l'esecuzione delle operazioni effettive.

java.util.StringTokenizer:

- **StringTokenizer**
  Divide una stringa in token (usando i delimitatori)
  Vuoto è il delimitatore predefinito
  Non distingue identificatori, numeri, commenti, stringhe tra virgolette

- **StreamTokenizer**
  Divide uno stream in token
  Più sofisticato, riconosce identificatori, commenti, stringa tra virgolette, numeri
  Usa la tabella dei simboli e le bandiere

MODULO 11

1. THREADS, PROCESSI, SCHEDULER E JVM

JVM E SISTEMA OPERATIVO:

a) Un sistema operativo multitasking assegna il tempo della CPU (sezioni) ai thread → Piccoli intervalli di tempo (20 ms) forniscono l'illusione del parallelismo (sulle macchine multi-core è un'illusione parziale)

Il sistema operativo è preventivo, se un thread viene eseguito fino a che:
- l'intervallo di tempo è scaduto o termina la sua esecuzione
- blocca (sincronizzazione con thread o risorse) - un altro thread acquisisce più priorità

b) JVM ottiene il CPU assegnato dal meccanismo di programmazione di OS

Java è una specifica con con molte diverse implementazioni*:

- Alcune JVM funzionano come un mini-OS e programmano i propri thread
- **La maggior parte delle JVM utilizza lo scheduler del sistema operativo (un thread Java è in realtà mappato a un thread di sistema)**

Processi:
Nei sistemi operativi (SO), ***un processo è un'istanza di un'applicazione in esecuzione***
→ Un processo ha il suo spazio indirizzo privato, codice, dati, file aperti, ecc.
• ***I processi non condividono la memoria*** (spazi di indirizzi separati), quindi devono comunicare attraverso meccanismi IPC offerti dai sistemi operativi (ad esempio, pipe, segnali)
• ***Un processo può contenere uno o più thread*** in esecuzione nel contesto del processo.

2. THREADS CONCETTI GENERALI

Ogni programma Java deve avere almeno un thread, il thread principale. Quando il programma inizia a funzionare, la JVM crea questo thread e chiama il metodo main() all'interno di quel thread.
→ Esistono altri thread creati da JVM che gli utenti di solito non notano (ad es. Garbage Collector).

Ragioni per usare i thread nei tuoi programmi Java:
Se usi la tecnologia Android, Swing, JavaFX, Servlet, RMI o Enterprise JavaBeans (EJB), potresti già utilizzare i thread senza accorgertene.
• Principali motivi per l'utilizzo dei thread:
- rendere l'interfaccia utente più reattiva
- sfruttare i sistemi multiprocessore
- eseguire operazioni asincrone o elaborazione in background

a) Thread singolo scenario:

In un ambiente di runtime a thread singolo, le azioni vengono eseguite una dopo l'altra

- Un'azione si verifica solo quando la precedente è stata completata.

b) Thread scenario multiplo:

- il download può essere eseguito in background (ad es. In un'altra discussione)
- anche l'analisi può essere eseguita in background (ovvero in un altro thread)
- ... tutto mentre l'utente interagisce con l'applicazione e attende eventuali notifiche (ricordi il tuo telefono?)

Vantaggi:

1. ***Abilita il parallelismo***
2. ***Più leggero dei processi per entrambi***

- Creazione (ovvero fork ()
- Comunicazione (ovvero tubi r / w ...)

Svantaggi:

1. Difficile per la maggior parte dei programmatori→ Anche per gli esperti, lo ***sviluppo è doloroso***
2. ***Le discussioni interrompono l'astrazione*:** non è possibile progettare moduli in modo indipendente.

3. THREADS CREAZIONE AVVIO TERMINAZIONE

In Java non è possibile chiamare esplicitamente il syscall fork () come in C.

Come avviene la creazione?

Le discussioni possono essere create da:

- estendere la classe Thread e sovrascriverne RUN() metodo.

- ***passare al costruttore della classe Thread un oggetto implementando l'interfaccia eseguibile.***

• È legale creare molti thread utilizzando lo stesso oggetto Runnable della destinazione.

a) Avviare thread:

Quando viene creato un oggetto Thread, ***non inizia l'esecuzione fino a quando non viene invocato il suo metodo start ().*** Quando ***un oggetto Thread esiste ma non è stato avviato***, ***è nel nuovo stato e non è considerato vivo.***

→ Il metodo start () può essere chiamato su un oggetto Thread una sola volta. Se start () viene chiamato più di una volta sullo stesso oggetto, genererà una RuntimeException

OSS:

• Non è garantito che i thread vengano eseguiti nello stesso ordine in cui sono stati chiamati i loro metodi start ().

• Non è garantito che un thread continui a essere eseguito fino al termine (non è garantito che il suo ciclo venga completato prima dell'inizio di un altro thread)

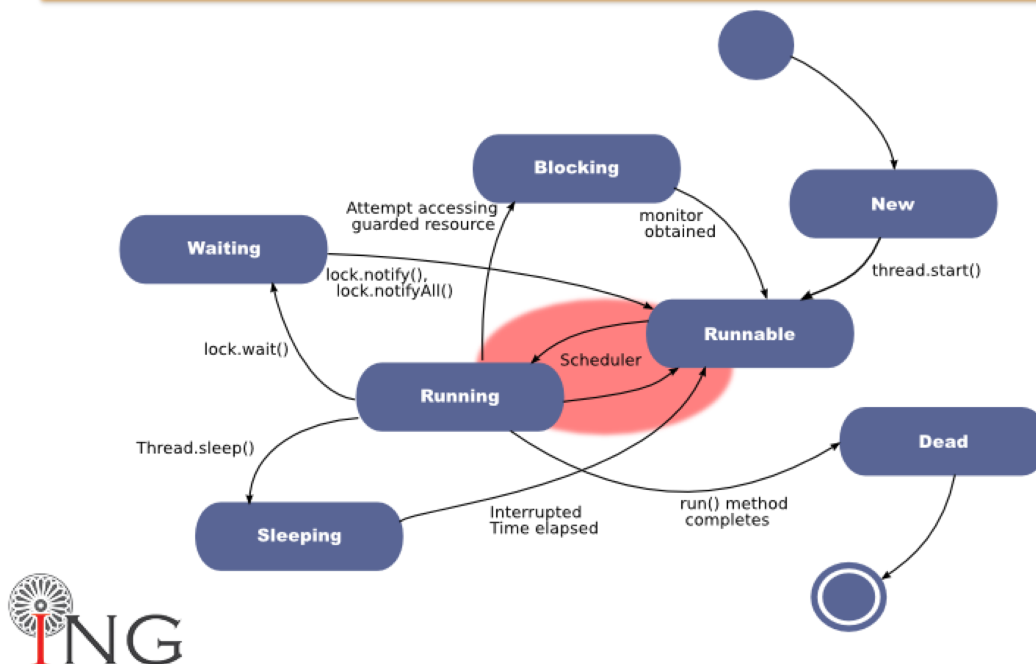
• Nulla è garantito tranne:

- Ogni thread verrà avviato e verrà eseguito fino al completamento dopo aver acquisito la CPU un numero finito di volte (si spera)

b) Terminare thread:

Se il thread padre termina, terminano anche tutti i thread figlio. Threaddi figlio condividono risorse con thread relativo, comprese le variabili. → Quando termina il thread padre, i thread figlio non saranno in grado di accedere a quelle risorse che il thread padre possiede. → Pertanto, se il genitore del lettore si dirige verso i suoi thread secondari, ***sono necessari meccanismi di sincronizzazione.***

4. THREADS STATI E CICLO DI VITA



1. Stato RUNNING:

Questo è lo stato in cui si trova un thread quando lo scheduler del thread lo seleziona (dal pool eseguibile) come thread attualmente in esecuzione.

2. Stato RUNNABLE:

Un thread che può essere eseguito, ma lo scheduler non lo ha selezionato come thread in esecuzione

A) Un thread prima entra nello stato eseguibile quando viene invocato il metodo start ()

B) Un thread può anche tornare allo stato eseguibile dopo lo stato in esecuzione, bloccato, in attesa o inattivo

• ***Quando un thread è nello stato eseguibile (RUNNABLE) , è considerato vivo***

3. Stato WAITING:

Un thread che può acquisire una risorsa ma non c'è lavoro da fare → Il thread chiama object.wait () e attende un altro thread che chiama object.notify () o object.notifyAll ()

4. Stato BLOCKING:

Un thread NON idoneo per l'esecuzione
Un thread bloccato in attesa di una risorsa (I / O o il blocco di un oggetto) ad es .:
- se i dati diventano disponibili in un inputstream da cui sta leggendo il thread
- Se il blocco di un oggetto diventa disponibile

5. Stato SLEEPING:

Un thread inattivo dopo una chiamata esplicita al metodo sleep()
→ Torna allo stato Runnable quando il thread si riattiva perché il tempo di sospensione è scaduto.

Oss: come lasciare lo stato running:

ci sono 3 modi per farlo:

- **Sleep():** il thread attualmente in esecuzione interrompe l'esecuzione per almeno la durata del sonno specificata

- **Yield():** il thread attualmente in esecuzione torna a runnable per dare spazio ad altri thread
  → *- Permette ad altri thread di fare il loro turno*
  - Tuttavia, potrebbe non avere alcun effetto: non vi è alcuna garanzia che il thread di cedimento non verrà pianificato di nuovo per l'esecuzione.

- **Join():** il thread attualmente in esecuzione si interrompe fino al completamento del thread che sta eseguendo il join
  → Un thread può eseguire un join thread per attendere fino al termine dell'altro thread. ***In generale, join thread è per un genitore l'unirsi con uno dei suoi thread figlio***.

PRIORITA’ THREAD:

Per default, un thread ottiene la priorità del thread che lo crea.
→ valori priorità sono stati definiti tra 1 e 10:

a) Thread.MIN_PRIORITY (== 1)
b) Thread.NORM_PRIORITY (== 5)
c) Thread.MAX_PRIORITY (== 10)

Un thread viene sempre eseguito con un numero di priorità

Criterio di pianificazione JVM:

Lo scheduler nella maggior parte delle JVM utilizza time-sliced (fettine di tempo), programmazione preventiva basata su priorità:

- a ogni thread viene assegnato un certo lasso di tempo, dopodiché viene rinviato a runnable per dare una possibilità a un altro thread

Le specifiche JVM non richiedono una VM per implementare uno scheduler time-slicing:

- alcune JVM possono utilizzare uno scheduler che consente a un thread di rimanere in esecuzione fino a quando non completa il suo metodo run().

OSS → CONSIGLI:

- *Alcuni metodi possono sembrare che dicono a un altro thread di bloccare, ma non lo fanno.*

- *Se t è un riferimento all'oggetto thread, è possibile scrivere qualcosa del tipo:* ***t.sleep () o t.yield ()***

  → ***Tuttavia, sono metodi statici della classe Thread:***
  non influenzano l'istanza t !!!
  invece influenzano il thread che è attualmente in esecuzione

- ***L'uso di una variabile di istanza per accedere a un metodo statico è soggetto a errori***

MODULO 12

1. SYNCRONIZED E LOCKING

PARTIAMO CON UN ESEMPIO:

Immagina due persone (rappresentate da due thread) ognuna con una carta bancomat collegata allo stesso account

```
class Account {
  private int balance;
  public int getBalance() {
     return balance;
  }

public void withdraw(int amount) {
balance -= amount;
    }
}
```

→ Ogni persona (ovvero thread) esegue questi passaggi:
1. Decidi un importo da prelevare
2. Controlla il saldo del conto.
3. Se ci sono abbastanza soldi, prelevare l'importo deciso
→ Cosa succede se lo scheduler sospende un thread tra il passaggio 2 e il passaggio 3 e l'altro viene eseguito?

SI HA UN PROBLEMA DI RACE CONDITION:

Un problema si verifica ogni volta che:
- Due o più thread condividono la stessa risorsa (in genere la variabile di istanza di un oggetto)
- Questo può produrre dati corrotti se un thread "si avvia" troppo rapidamente prima che un'operazione sia stata completata.

→ COME PREVENIRE RACE CONDITIONS:

- ***Dobbiamo garantire che le due fasi del ritiro non siano MAI divise***

- ***Il prelievo deve essere un'operazione atomica:***
  Qualsiasi prelievo deve essere completato prima di qualsiasi l'altro thread è autorizzato ad agire sull'account

  Indipendentemente dal numero di istruzioni effettive!

SINCRONIZZAZIONE:
Non puoi garantire che ti stai muovendo durante una intera operazione (si suppone che sia atomico per evitare le condizioni di gara).
• **Gli sviluppatori non possono controllare lo scheduler** (escluso il caso di chiamare yield ()).
• Il ***modificatore sincronizzato può essere applicato a un metodo o un oggetto.***
• La sincronizzazione blocca un blocco di codice: *solo un thread alla volta può accedere.*

2. ARCHITETTURE PRINCIPALI

SINCRONIZZAZIONE E LOCKS (BLOCCHI):

Ogni oggetto in Java ha un blocco integrato
• Inserire un metodo non statico sincronizzato significa ottenere il blocco dell'oggetto. Se un thread ottiene il blocco, gli altri thread devono attendere per immettere il codice sincronizzato fino al rilascio del blocco (il thread esce dal metodo sincronizzato)
• Non tutti i metodi in una classe devono essere sincronizzati.
→ ***Una volta che un thread ottiene il blocco su un oggetto, nessun altro thread può entrare in nessuno dei metodi sincronizzati dell'oggetto.***

• Più thread possono ancora accedere ai metodi non sincronizzati della classe
- I metodi che non accedono ai dati critici non devono essere sincronizzati
- I thread che vanno a dormire non rilasciano i blocchi!
• Un thread può acquisire più di un blocco
- Ad esempio, un thread può inserire un metodo sincronizzato, quindi richiamare immediatamente un metodo sincronizzato su un altro oggetto (soggetto a deadlock!)

Codice sincronizzato
• Il problema più comune dell'accesso concorrente è il problema produttore-consumatore
- Il thread del produttore inserisce gli elementi in un oggetto condiviso. Il thread consumer li recupera
• Esistono diversi modi per sincronizzare l'accesso a un oggetto condiviso. Fondamentalmente:
- Usa le classi thread-safe come oggetti condivisi (usano sincronizzati sui loro metodi)

- Utilizzare la sincronizzazione in produttori e consumatori per bloccare l'oggetto condiviso

CLASSE THREAD-SAFE:

Una classe thread-safe è una classe sicura (funziona correttamente) quando si accede da più thread. Le ***sezioni critiche*** (cioè le sezioni contenenti le condizioni di gara) ***sono incapsulate in metodi sincronizzati.***

- ArrayList non è sicuro!
- Vector è un equivalente thread-safe di ArrayList (tutti i metodi sono sincronizzati)

3. SYNCRONIZED, WAIT, NOTIFY

SINCRONIZZAZIONE USANDO OBJECT:

Produttori e consumatori potrebbero essere in grado di bloccare (acquisire accesso esclusivo) una risorsa condivisa ma non essere ancora in grado di progredire

→ Per evitare uno spreco di risorse possiamo usare: - yield ()
- wait () / notification ()

WAIT:

Il metodo wait () è un metodo di istanza utilizzato per la sincronizzazione dei thread.

- wait () ***può essere chiamato su qualsiasi oggetto***, come è definito direttamente su java.lang.Object
- wait () ***può essere chiamato solo da un blocco sincronizzato***. Rilascia il blocco sull'oggetto in modo che un altro thread possa saltare e acquisire un blocco.

→ **Void wait():** *Fa in modo che il thread corrente attenda fino a quando un altro thread invoca il metodo notification () o il metodo notifyAll () per questo oggetto.*

METODO NOTIFY:

Il metodo notify () invia un segnale a uno dei thread in attesa nel pool di attesa dello stesso oggetto.

1. Il metodo notify () NON PUO' specificare quale thread in attesa di notificare.
   → **Void notify():** *Riattiva un singolo thread in attesa sul blocco dell'oggetto.*

2. Il metodo notifyAll () è simile a notify() ma invia un segnale a tutti i thread in attesa sull'oggetto. → **Void notifyAll():** *Riattiva tutti i thread in attesa sul blocco di questo oggetto.*

OSS: PROBLEMI THREAD

Se il codice è sincronizzato correttamente, potrebbe non funzionare.
I problemi principali sono:
-Deadlock (attesa indefinita): ***si verifica quando due thread sono bloccati, uno in attesa del blocco dell'altro. Nessuno dei due può correre finché l'altro non abbandona il lock, quindi aspettano per sempre.***

→ • Una progettazione inadeguata può portare a un deadlock
- È difficile eseguire il debug del codice per evitare deadlock
- Il controllo del modello potrebbe essere una soluzione (problema: esplosione dello spazio degli stati)

- **Livelock** (thread in esecuzione ma non viene eseguito alcun lavoro): ***Un thread agisce spesso in risposta all'azione di un altro thread.*** Se anche l'azione dell'altro thread è una risposta all'azione di un altro thread, potrebbe verificarsi un livelock.

→ • Come per il deadlock, i thread con livelock non sono in grado di fare ulteriori progressi.
• Tuttavia, i thread non sono bloccati: sono semplicemente troppo impegnati a rispondere l'un l'altro per riprendere il lavoro.

- **Starvation** (thread non viene mai eseguito): ***descrive una situazione in cui un thread non è in grado di ottenere un accesso regolare alle risorse condivise e non è in grado di fare progressi.***

→ • Ad esempio, supponiamo che un oggetto fornisca un metodo sincronizzato che spesso richiede molto tempo per tornare. Se un thread invoca questo metodo frequentemente, altri thread che necessitano anche di un frequente accesso sincronizzato allo stesso oggetto verranno spesso bloccati.

IN CONCLUSIONE:

Sincronizzazione

- Deve coordinare l'accesso ai dati condivisi con i blocchi. Hai dimenticato un lock? Goditi i dati corrotti.

Problemi di prestazione

- Il blocco semplice produce una bassa concorrenza.
- Il bloccaggio a grana fine aumenta la complessità

Difficile da eseguire il debug

- Dipendenze di dati e temporizzazione
- Pochi strumenti di debug