Relazione tecnica - Progetto Bacheca Annunci

Documento tecnico che illustra le scelte progettuali, le classi, le loro relazioni e i metodi pubblici (parametri, valori di ritorno e eccezioni) del progetto.

Sommario esecutivo — scelte architetturali

- **Pattern architetturale**: MVC (Model–View–Controller).
- **Model** → gestione dei dati e delle regole di business: Annuncio, Bacheca, Utente, eccezioni custom (AnnuncioException, BachecaException, UtenteException, AutoreNonAutorizzatoException).
- **View** → interfacce utente: interfaccia.grafica (GUI Swing) e interfaccia.rigaDiComando (CLI).
- **Controller** → interfaccia.grafica.controllo coordina le view e applica le regole di business tramite la Bacheca.
- **Persistenza** → file testuale semplice (annunci.txt) gestito da Bacheca mediante serializzazione/parse testuale per scopi didattici.
- **Testing** → JUnit 5, con test unitari per Utente, Annuncio e Bacheca.
- **Design chiave**:
    - Semplicità e chiarezza del flusso applicativo.
    - Separazione delle responsabilità (UI separata dal modello).
    - Validazione centralizzata nel modello per coerenza dei dati.
    - Gestione di eccezioni custom per distinguere errori di dominio (validazione, autorizzazione) da errori di I/O.

Relazioni principali tra classi

- Bacheca contiene una collezione (List) di oggetti Annuncio.
- Annuncio mantiene un riferimento all'Utente autore.
- ControlloBacheca opera su una Bacheca e viene utilizzato sia dalla GUI sia dalla CLI per applicare le operazioni principali: aggiunta, rimozione, ricerca, pulizia annunci scaduti.
- Le view (BachecaPanel, ContentPanel, OpsPanel, UtentePanel) visualizzano i dati e invocano i metodi di ControlloBacheca o direttamente della Bacheca tramite il controller.

- La CLI (InterfacciaRigaDiComando) interagisce direttamente con il modello (Bacheca) e utilizza Scanner per input testuale, gestendo il flusso di login, menu e operazioni.

Documentazione delle classi principali:

modello.Annuncio

Pacchetto: modello
Ruolo: rappresenta un annuncio pubblicato nella bacheca.

Campi principali

- private final int **id** — identificativo univoco (intero positivo).
- private final Utente **autore** — autore dell'annuncio (non null).
- private final String **articolo** — nome/sintesi dell'articolo (non null/non blank).
- private final float **prezzo** — prezzo (deve essere > 0.0f).
- private final String **tipologia** — tipologia normalizzata ("acquisto" o "vendita", sempre lowercase).
- private final Set<String> **paroleChiave** — LinkedHashSet per mantenere ordine d'inserimento ed evitare duplicati; può essere vuoto.
- private final LocalDate **dataScadenza** — data di scadenza per le vendita; null per acquisto.

Costruttori

- public Annuncio(int id, Utente autore, String articolo, float prezzo, String tipologia, String paroleChiave, String dataScadenza) throws AnnuncioException
  Validazioni:
  - **autore** non può essere **null**.
  - **articolo** non può essere **null o vuoto**.
  - **prezzo** deve essere > **0.0f** (altrimenti AnnuncioException).
  - **tipologia** non può essere **null** ed è normalizzata in lowercase; deve essere "acquisto" o "vendita".

    - **paroleChiave** (se non null/vuoto) deve rispettare il pattern CSV definito da PAROLE_CHIAVE_REGEX; le singole parole sono trimmate e validate.
    - **dataScadenza** è obbligatoria per vendita (deve rispettare yyyy-MM-dd e parsarsi in LocalDate), mentre per acquisto viene impostata a null.
    - In caso di violazione viene lanciata **AnnuncioException** con messaggio esplicito.
- public Annuncio(Utente autore, String articolo, float prezzo, String tipologia, String paroleChiave, String dataScadenza) throws AnnuncioException
    - Convenienza: invoca il costruttore principale con un id generato da generaId().

Metodi pubblici principali

- **public boolean isScaduto()**
    - Ritorna **true** se **dataScadenza != null** e **dataScadenza.isBefore(LocalDate.now())**. Per **acquisto** (data **null**) ritorna false.
- **public void aggiungiParola(String nuovaParola) throws AnnuncioException**
    - Valida **nuovaParola** (non **null**, non vuota, match con **PAROLA_SINGOLA_REGEX**) e la aggiunge al **Set** interno. Lancia **AnnuncioException** se non valida.
- **public int getId()** — ritorna **id**.
- **public Utente getAutore()** — ritorna l'autore (riferimento immutabile come nel codice).
- **public String getArticolo()** — ritorna il campo **articolo**.
- **public float getPrezzo()** — ritorna il prezzo; attenzione alle comparazioni a causa del tipo **float** (usare delta nei test).
- **public String getTipologia()** — ritorna la tipologia normalizzata ("**acquisto**" o "**vendita**").

- **public String getParoleChiave()** — ritorna le parole chiave come CSV con separatore **", "** (es. "**elettronica, TV**"). Questo è stato scelto per compatibilità con il formatting usato in **toString()** e con il parsing semplice della **Bacheca**.

---

modello.Bacheca

Pacchetto: modello
Ruolo: gestisce la collezione di Annuncio, la validazione semplice degli inserimenti/rimozioni e la persistenza testuale su file.

Campi principali

- **private final List<Annuncio> annunci** — lista degli annunci (ordine di inserimento).
- **private final Set<Integer> poolId** — pool di id esistenti (evita duplicati, **contains** O(1)).

Costruttore

- **public Bacheca()** — inizializza la lista annunci e tipicamente carica da file annunci.txt se presente.

Metodi pubblici principali

- **public Iterator<Annuncio> iterator()**
    - **Ritorno**: Iterator<Annuncio> su Collections.unmodifiableList (annunci).
    - **Scopo:** permettere l'iterazione sicura (senza rimozione tramite l'iterator).
- **public ArrayList<Annuncio> aggiungiAnnuncio(Annuncio annuncio) throws BachecaException**
    - **Parametri:** a (Annuncio) — annuncio da aggiungere.
    - **Ritorno: ArrayList<Annuncio>** — se l'annuncio è di tipo **acquisto** ritorna la lista degli annunci di **vendita** compatibili per parole chiave; altrimenti lista vuota.
    - **Eccezioni: BachecaException** se annuncio nullo, id già presente o annuncio già presente.

    - **Scopo:** aggiunge l'annuncio e aggiorna **poolId**; difende dall'inserimento di duplicati.
- **public boolean rimuoviAnnuncio(int id, Utente utente) throws AutoreNonAutorizzatoException, BachecaException.**
    - **Parametri:** id dell'annuncio; utente che richiede la rimozione.
    - **Ritorno:** true se rimosso con successo.
    - **Eccezioni:** AutoreNonAutorizzatoException se il richiedente non è l'autore; BachecaException se annuncio non trovato.
    - **Scopo:** rimozione controllata con autorizzazione e aggiornamento poolId.
- **public ArrayList<Annuncio> cercaPerParolaChiave(String paroleChiave)**
    - **Parametri:** stringa CSV di parole chiave (es. "elettronica, TV").
    - **Ritorno:** lista di annunci che hanno almeno una parola chiave in comune (confronto case-insensitive, trimmed).
    - **Scopo:** ricerca semantica basata su intersezione di parole chiave.
- **public boolean pulisciBacheca()**
    - **Ritorno:** true se è stato rimosso almeno un annuncio scaduto; false altrimenti.
    - **Scopo:** rimuove gli annunci per i quali Annuncio.isScaduto() è true e pulisce poolId.
- **public boolean aggiungiNuovaParolaChiave(int id, Utente utente, String nuovaParola) throws AutoreNonAutorizzatoException, AnnuncioException**
    - **Parametri:** id annuncio, utente richiedente, nuovaParola.
    - **Ritorno:** true se la parola è stata aggiunta.
    - **Eccezioni:** AutoreNonAutorizzatoException se non è autore; AnnuncioException se annuncio non trovato o parola non valida.
    - **Scopo:** autorizzare e delegare ad Annuncio.aggiungiParola(...).
- **public List getAnnunci()**

    - **Ritorno:** copia dell'elenco annunci (modificabile). Utile per test; evita l'esposizione diretta della lista interna.

- **public Set<Integer> getPoolId()**
    - **Ritorno:** riferimento al poolId (modificabile nel codice attuale; usato nei test).

Metodi I/O (caricamento/salvataggio)

- **public void salvaAnnunciSuFile(String fileName)**
    - Salva tutti gli annunci correnti su file di testo.
    - **Eccezioni**: IOException in caso di errore di scrittura.
- **public void caricaAnnunciDaFile(String fileName)**
    - Carica gli annunci da file, svuotando prima la bacheca e resettando il poolId.

---

modello.Utente

Pacchetto: modello
Ruolo: rappresenta un utente registrato o connesso.

Campi principali

- private final String **email** — indirizzo email dell'utente (immutabile).
- private final String **nome** — nome dell'utente (immutabile).
- private final String **emailRegex** — regex per validare il formato dell'email.
- private static final String **nomeRegex** — regex per validare il formato del nome.

Costruttore

- **public Utente(String email, String nome)**
    - **Parametri**: email (String), nome (String).
    - **Eccezioni**: UtenteException se email o nome non rispettano i formati richiesti.
    - **Motivo**: garantire la correttezza dei dati all'atto della creazione.

Metodi pubblici principali

- **public String getEmail()** → restituisce l'email.
- **public String getNome()** → restituisce il nome.
- **@Override public String toString()** → restituisce una rappresentazione testuale dell'utente nel formato [email= ..., nome= ...].
- **@Override public boolean equals(Object obj)** → due utenti sono uguali se hanno stessa email e stesso nome.
- **@Override public int hashCode()** → coerente con equals().

Metodi privati di supporto

- **private boolean isEmailValida(String email)** → controlla il formato email con regex.
- **private boolean isNomeValido(String nome)** → controlla il formato nome con regex.

Eccezioni nel package modello.exception

AnnuncioException

- **Quando:** errori di validazione o integrità di un annuncio (titolo vuoto, prezzo negativo, formato parole chiave non valido).
- **Perché:** permette di distinguere errori di dominio legati agli annunci dagli errori generici del sistema.
- **Implementazione:** estende Exception, costruttore con messaggio descrittivo.

AutoreNonAutorizzatoException

- **Quando:** un utente tenta di rimuovere o modificare un annuncio che non è stato da lui creato.
- **Perché:** implementare controllo di autorizzazione direttamente nel modello o nel controller.
- **Implementazione:** estende Exception, costruttore con messaggio descrittivo.

BachecaException

- **Quando:** errori legati alla gestione della bacheca, tipicamente problemi di I/O (lettura/scrittura su file) o parsing malformato.
- **Perché:** separare errori infrastrutturali da errori di dominio.
- **Implementazione:** estende Exception, costruttore con messaggio descrittivo.

UtenteException

- **Quando:** problemi di creazione o validazione dell'utente (email non valida, nome mancante o errato).
- **Perché:** centralizzare la gestione degli errori di validazione utente.
- **Implementazione:** estende Exception, con due costruttori:
  - uno con messaggio descrittivo,
  - uno con messaggio e causa dell'errore.

---

interfaccia.grafica.InterfacciaGrafica

Pacchetto: interfaccia.grafica
Ruolo: costruisce e mostra l'interfaccia Swing principale della bacheca annunci.

Campi principali

- private Bacheca model — riferimento al modello della bacheca.
- private Utente utente — utente corrente autenticato tramite dialog di login.

Costruttore

- **public InterfacciaGrafica()**
  - Inizializza la bacheca (model).
  - Avvia il login dell'utente con input GUI e sanitizzazione dei dati.
  - Carica gli annunci da file.
  - Configura la finestra (JFrame), imposta pannello principale (BachecaPanel), dimensioni e visibilità.

Metodi principali

- **private void caricaBacheca()**
    - Carica gli annunci da annunci.txt.
    - In caso di errore mostra un messaggio e termina il programma.
- **private void LogIn()**
    - Mostra dialog con campi nome/email.
    - Sanitizza l'input (rimozione caratteri invisibili, normalizzazione).
    - Crea un nuovo oggetto Utente.
    - Se la validazione fallisce, mostra messaggio di errore e ripete l'input.
- **private static String sanitizeString(String s)**
    - Normalizza e pulisce l'input del nome (Unicode NFC, rimozione spazi invisibili, caratteri di controllo, trim).
- **private static String sanitizeEmail(String email)**
    - Normalizza e pulisce l'input email (trim, rimozione caratteri di controllo e NBSP).
- **private static void debugPrintCodepoints(String s, String label)**
    - Stampa a console i codepoint Unicode di una stringa per debugging.

Entry point

- **public static void main(String[] args)** *(non incluso ma tipico)* → permette di avviare direttamente la GUI.

---

interfaccia.grafica.controllo.ControlloBacheca

Pacchetto: interfaccia.grafica.controllo
Ruolo: Controller che gestisce le operazioni sulla bacheca, facendo da ponte tra vista (**ContentPanel**) e modello (**Bacheca**). Riceve le azioni dell'utente, le valida ed esegue le operazioni sul model aggiornando la view.

Costruttore

- **public ControlloBacheca(ContentPanel view, Bacheca model, Utente utente)**

- Parametri: la vista associata, il modello Bacheca e l’utente loggato.

Metodi principali (azioni utente)

- **private void aggiungi()**
  - Permette di creare un nuovo annuncio tramite dialogo, validarlo e salvarlo su file.
  - Mostra eventuali annunci compatibili in caso di tipologia "acquisto".
- **private void rimuovi()**
  - Rimuove un annuncio identificato da ID (controllando l’autore).
  - Aggiorna il file degli annunci.
- **private void cerca()**
  - Ricerca annunci per parola chiave, mostrando i risultati con evidenziazione grafica.
- **private void pulisci()**
  - Rimuove automaticamente gli annunci scaduti e aggiorna il file.
- **private void aggiungiParolachiave()**
  - Consente di aggiungere nuove parole chiave a un annuncio esistente (autorizzato).

Metodi tecnici

- **public void actionPerformed(ActionEvent e)**
  - Gestisce le azioni sui pulsanti e richiama il metodo corrispondente.
- **public List<Annuncio> cercaAnnunci(String keyword)**
  - Metodo previsto ma non ancora implementato (**UnsupportedOperationException**).

Pannelli GUI (package interfaccia.grafica.vista)

Queste classi costituiscono la **view** e forniscono componenti modulari per l’interfaccia utente della bacheca.

BachecaPanel

- **Scopo:** pannello principale che compone la GUI della bacheca (titolo, operazioni, lista annunci, info utente).
- **Costruttore:**
    - public BachecaPanel(Bacheca model, Utente utente) → costruisce il layout integrando ContentPanel, OpsPanel, UtentePanel.

ContentPanel

- **Scopo:** mostra dinamicamente la lista degli annunci.
- **Costruttore:**
    - public ContentPanel(Bacheca model) → inizializza e avvia aggiornamento periodico automatico.
- **Metodi pubblici:**
    - public void updateDisplay() → ricostruisce e ridisegna la lista annunci.

OpsPanel

- **Scopo:** contiene i pulsanti delle operazioni disponibili sulla bacheca (aggiungi, rimuovi, cerca, pulisci, aggiungi parole chiave).
- **Costruttore:**
    - public OpsPanel(ControlloBacheca controllo) → crea i pulsanti e li collega al controller.

UtentePanel

- **Scopo:** mostra le informazioni dell'utente corrente (nome, email, icona).
- **Costruttore:**
    - public UtentePanel(Utente utente, ControlloBacheca controllo) → inizializza il pannello footer utente.

InterfacciaRigaDiComando

Pacchetto: interfaccia.rigaDiComando
Ruolo: Fornisce una CLI testuale per interagire con la **Bacheca**. Utile per debug o utilizzo senza GUI.

Costruttore

- **public InterfacciaRigaDiComando(Scanner scanner)**
    - Inizializza la bacheca, prova a caricare gli annunci da file (**annunci.txt**), gestisce il login utente e avvia il ciclo principale.

Metodi principali

- **private void caricaBacheca()**
    - Carica gli annunci da file; in caso di errore prosegue con bacheca vuota.
- **private void logIn()**
    - Richiede nome ed email, crea un oggetto **Utente**; ripete fino a validazione corretta.
- **private void run()**
    - Ciclo principale: mostra menu, legge input, inoltra le richieste ai metodi dedicati.
- **private void aggiungiAnnuncio()**
    - Crea un annuncio, valida i campi, lo aggiunge alla bacheca e salva su file.
- **private void rimuoviAnnuncio()**
    - Rimuove un annuncio se l'utente è autore, con controllo autorizzazione.
- **private void cercaAnnuncio()**
    - Ricerca annunci per parole chiave e stampa i risultati.
- **private void pulisciBacheca()**
    - Rimuove annunci scaduti e salva i cambiamenti.
- **private void visualizzaBacheca()**
    - Mostra tutti gli annunci presenti.
- **private void aggiungiNuovaParolaChiave()**
    - Consente di aggiungere nuove parole chiave a un annuncio, con validazione e salvataggio.

---

main.Main

Pacchetto: main
Ruolo: Punto di ingresso dell'applicazione. Permette all'utente di scegliere se avviare la modalità **grafica (Swing)** o la **riga di comando (CLI)**.

Metodi

- public static void main(String[] args)
    - **Descrizione:** avvia il programma, mostra il menu, legge la scelta da tastiera e avvia la modalità selezionata.
    - **Parametri:** args — argomenti opzionali da riga di comando (non utilizzati).
    - **Eccezioni:** gestisce eccezioni generiche derivanti da GUI o CLI, loggandole a console.
- private static void stampaMenu()
    - **Descrizione:** stampa a console il menu principale con le opzioni disponibili.
    - **Ritorno:** void.

Test (package modello.test)

Pacchetto: modello.test

Ruolo: test di unità per le classi del modello (Annuncio, Bacheca, Utente).

Classi di test

1. AnnuncioTest

Testa la creazione e il comportamento degli oggetti Annuncio (parole chiave, scadenza, prezzo, tipologia).

Metodi principali di test:

- **testCostruttoreConVendita()** – verifica creazione annuncio di tipo vendita con data di scadenza.
- **testCostruttoreConVenditaConId()** – verifica creazione annuncio con ID esplicito.
- **testCostruttoreConAcquisto()** – verifica creazione annuncio di tipo acquisto senza scadenza.
- **testDataScadenzaNonValida()** – controlla gestione di date scadute o formati errati.
- **testPrezzoNonValido()** – controlla che prezzo negativo generi eccezione.
- **testTipologiaNonValida()** – verifica eccezioni per tipologie diverse da "acquisto" o "vendita".
- **testParoleChiaveNonValide()** – verifica formato corretto delle parole chiave.

---

2. BachecaTest

Testa le operazioni principali della bacheca: aggiunta, rimozione, ricerca e pulizia annunci.

Metodi principali di test:

- **testAggiungiAnnuncioVendita()** – aggiunta annuncio valido.
- **testAggiungiAnnuncioDuplicato()** – verifica eccezione su duplicati.
- **testAggiungiAnnuncioStessoID()** – controllo id univoci.
- **testRimuoviAnnuncio()** – rimozione annuncio da utente autorizzato.
- **testRimuoviAnnuncioAutoreNonAutorizzato()** – eccezione per utente non autorizzato.
- **testRimuoviAnnuncioNonTrovato()** – eccezione se annuncio inesistente.
- **testCercaPerParolaChiave()** – ricerca annunci per parole chiave.
- **testPulisciBacheca()** – rimozione annunci scaduti.
- **testAggiungiNuovaParolaChiave()** – aggiunta di parole chiave a un annuncio.

- **testCaricaSalvaDaFile()** – verifica persistenza su file e ricaricamento.
- **testRimuoviAnnuncioConIteratore()** – verifica eccezione su iteratore non modificabile.

---

3. UtenteTest

Verifica la validazione degli oggetti Utente (email e nome).

Metodi principali di test:

- **testCostruttoreValido()** – creazione utente con dati corretti.
- **testCostruttoreEmailNonValida()** – email errata genera eccezione.
- **testCostruttoreNomeNonValido()** – nome non alfanumerico genera eccezione.

Commento di progetto:

I test coprono sia i **casi validi** sia i **casi eccezionali**, garantendo la correttezza della logica di business e la robustezza delle classi modello. I test di Bacheca includono anche la persistenza su file, la gestione degli ID unici e la protezione contro modifiche tramite iteratori.

Scelte di progettazione e motivazioni dettagliate

1. **Architettura MVC (Model-View-Controller)**
    - Il progetto separa chiaramente il **modello** (Bacheca, Annuncio, Utente) dalle **interfacce utente** (CLI e GUI).
    - La **view** non contiene logica di business, limitandosi a raccogliere input e mostrare output.
    - La **controller-like logic** è delegata al modello stesso (Bacheca) per operazioni come aggiunta, rimozione, ricerca e gestione parole chiave.
    - Questo approccio semplifica il testing unitario del modello senza dover simulare l'interfaccia utente e facilita la manutenzione e l'estendibilità (es. aggiungere nuove view o funzionalità).
2. **Uso di eccezioni custom**

- Sono state definite eccezioni dedicate per ciascun dominio:
  - **BachecaException** per errori generici di bacheca,
  - **AnnuncioException** per validazione degli annunci,
  - **UtenteException** per errori di creazione utente,
  - **AutoreNonAutorizzatoException** per accessi non consentiti.
- Questa distinzione consente alla CLI e alla GUI di gestire **errori di dominio** in modo chiaro e distinto dagli errori di **infrastruttura** (es. I/O con file).
- Nei test, permette di verificare il comportamento corretto del modello senza confondere eccezioni di tipo diverso.

3. **Persistenza testuale dei dati**
   - Gli annunci vengono salvati e caricati da un file di testo (annunci.txt) nella working directory.
   - Scelta motivata da semplicità didattica e leggibilità diretta dei dati.
   - In un contesto reale, si consiglierebbe l'uso di un database o di formati strutturati (JSON/XML) per maggiore robustezza, gestione concorrente e facilità di query avanzate.
4. **Centralizzazione delle regole nel modello**
   - Operazioni come aggiunta, rimozione, pulizia annunci scaduti, e aggiunta di parole chiave vengono gestite **interamente nel modello** (Bacheca).
   - Le interfacce (CLI o GUI) si limitano a raccogliere input e mostrare output, evitando duplicazione di logica di business o controlli di autorizzazione.
   - Questo approccio riduce errori di incoerenza tra diverse view e rende il comportamento del sistema prevedibile e consistente.
5. **Validazione centralizzata degli oggetti**
   - Ogni oggetto del modello verifica autonomamente la propria validità:
     - Utente controlla formato email e nome,

        - Annuncio controlla prezzo, tipologia, formato parole chiave e data di scadenza.
    - Le regole di validazione centralizzate nel modello impediscono che interfacce diverse o sviluppatori diversi bypassino le regole, mantenendo **coerenza dei dati** e semplificando i test unitari.

Possibili miglioramenti futuri

1. **Persistenza più robusta e strutturata**
    - Sostituire il salvataggio su file di testo con **JSON** o **SQLite** per:
        - maggiore leggibilità dei dati,
        - gestione di query più complesse (es. filtri avanzati per parole chiave, tipologia o scadenza),
        - riduzione del rischio di corruzione dati in caso di scritture simultanee.
2. **Logging centralizzato**
    - Introdurre un sistema di logging (es. java.util.logging o **SLF4J**) per tracciare:
        - errori imprevisti,
        - operazioni critiche come aggiunta/rimozione annunci,
        - tentativi di accesso non autorizzati.
    - Questo migliorerebbe il debug e il monitoraggio del sistema senza appesantire l'output delle interfacce.
3. **Protezione dei dati della bacheca**
    - Modificare Bacheca.getAnnunci() per restituire una **copia immutabile** degli annunci, evitando che il contenuto della bacheca possa essere modificato direttamente dalle view o da codice esterno.
4. **Autenticazione e autorizzazione più robuste**
    - Introdurre **sessioni utente**, autenticazione con **password hashed** e gestione dei permessi.

- Questo permetterebbe di:
  - proteggere le operazioni di modifica/rimozione annunci,
  - distinguere tra diversi livelli di utenza (es. admin vs utente standard) se il sistema dovesse crescere.

5. **Miglioramento dei test**
   - Ampliare i test unitari e di integrazione per:
     - coprire casi limite (es. titoli vuoti, date di scadenza al limite, prezzo zero),
     - simulare operazioni di I/O su file senza scrivere realmente sul disco (**mock file system**),
     - verificare correttamente la gestione degli errori nelle interfacce CLI e GUI.

Conclusione

La relazione fornisce una panoramica completa delle scelte progettuali, delle classi principali, dei metodi pubblici e della gestione delle eccezioni nel progetto **Bacheca Annunci**.

Per proseguire, si possono intraprendere diverse opzioni pratiche per migliorare o documentare ulteriormente il progetto:

1. **Generazione file markdown**
   - Creare automaticamente un file Relazione_BachecaAnnunci.md pronto da scaricare, contenente tutte le informazioni raccolte: classi, metodi, test, scelte progettuali e possibili miglioramenti.
2. **Schema del file annunci.txt**
   - Integrare la relazione con una sezione che descriva il formato e il pattern utilizzato in annunci.txt, basandosi direttamente sul codice di serializzazione della bacheca.
3. **Javadoc inline**
   - Inserire template di commenti Javadoc direttamente in ogni file .java, per rendere immediatamente consultabile la documentazione

dei costruttori, metodi e parametri senza aprire la relazione separata.

Queste opzioni permettono di:

- avere documentazione completa pronta per l'uso,
- facilitare manutenzione e sviluppo futuro,
- migliorare leggibilità e tracciabilità del progetto.