Università degli Studi di Milano-Bicocca

Corso di Laurea in Informatica

Processo e Sviluppo del Software

Progetto di Clean Code

Principi applicati al toolkit OpenMap

Asia Zakiah Piazza
Matricola: 899552

Anno Accademico 2025-2026

Indice

1. Introduzione al progetto

OpenMap è un toolkit open-source nato per creare applicazioni geografiche utilizzando la tecnologia Java. Grazie alla sua struttura flessibile, permette di visualizzare e gestire mappe e dati geospaziali in modo dinamico. Tuttavia, l'ultima versione ufficiale risale a più di dieci anni fa: questo significa che il codice potrebbe essere implementato con logiche datate.

L'obiettivo di questo progetto è effettuare un'operazione di refactoring a partire da 5 classi eterogenee, spiegandone le motivazioni sulla base dei principi del Clean code. A seguito, il codice verrà commentato (se necessario), indentato e formattato correttamente. Vengono brevemente descritte le 5 classi scelte:

- **AboutMenuItem**: componente grafico che estende gli elementi dei menu di Java Swing per mostrare le informazioni sul software in una sezione apposita.
- **DefaultOverviewMouseMode**: classe controller che definisce come il mouse interagisce con la finestra della panoramica su una mappa.
- **ImageServerUtils**: classe di utilità che fornisce metodi per la gestione delle immagini e dei colori.
- **ESRIRecord**: classe di entità che rappresenta l'header di ogni record geografico.
- **HelloWorldLayer**: classe di esempio per mostrare come visualizzare poligoni sulla mappa.

Il codice comprensivo delle classi originali e delle versioni post-refactoring è disponibile a seguito: https://github.com/asiiapiazza/2025_assignment3_clean_code
Repository ufficiale OpenMap: https://github.com/OpenMap-java/openmap

2. AboutMenuItem

La classe `AboutMenuItem` è un componente dell'interfaccia grafica che ha il compito di costruire e mostrare una finestra di dialogo contenente le informazioni sull'applicazione in una sezione "About". È composta da un costruttore e 4 metodi.

2.1 actionPerformed(ActionEvent ae)

Il metodo `actionPerformed` viene invocato quando l'utente clicca sulla voce di menu "About" con l'obiettivo di visualizzare una finestra pop-up. Nell'implementazione viene violato il principio della singola responsabilità (SRP) poiché il metodo gestisce contemporaneamente la logica di controllo dello stato di `aboutBox` e la logica di UI.

Viene quindi applicato un refactoring basato sull'estrazione dei metodi, per separare nettamente le responsabilità delegando ai nuovi metodi `initializeAboutBox`, `createAboutBox` e `setupLayout`. Inoltre si vuole evitare la ripetizione dell'uso del metodo `getContentPane()`. Vengono dunque definiti i seguenti metodi:

- `actionPerformed(ActionEvent event)`: il suo compito ora è quello di chiamare il metodo `initializeAboutBox` e rendere visibile la box se questa è già definita.
- `initializeAboutBox()`: gestisce il ciclo di vita della finestra di dialogo, garantendo che venga creata e configurata solo alla prima invocazione con i metodi ausiliare `createAboutBox` e `setupLayout`.
- `createAboutBox()`: metodo per la creazione della finestra di dialogo
- `setupLayout()`: si occupa della costruzione del layout assemblando i vari componenti grafici nelle giuste posizioni.

2.2 createCopyrightViewer()

Il metodo ha il compito di generare un componente grafico per la visualizzazione delle informazioni del software. Notiamo come la scelta del nome `createCopyrightViewer` non risulta corretta: infatti il metodo non implementa nessun pattern di tipo *Viewer*, violando il principio dei *meaningful names*. Inoltre, viene meno il principio *SRP*, poiché il metodo gestisce la logica di costruzione delle stringhe e la creazione dei componenti grafici necessari alla loro visualizzazione, come per il metodo precedente. Estraiamo quindi i seguenti metodi:

- `createCopyrightInfo()`: viene rinominato `createCopyrightViewer`. Il metodo funge da coordinatore richiamando il metodo `buildCopyrightText()` e restituendo l'oggetto `JScrollPane`.
- `buildCopyrightText()`: si occupa della generazione del testo composto dalla versione e dalla data della build. Sebbene sia possibile un'ulteriore scomposizione per costruire le due stringhe in metodi diversi, si è scelto di mantenere la logica integra poiché la sequenza di azioni è già sufficientemente chiara. La stringa finale viene rinominata `copyrightInformation`.
- `createScrollingTextArea(String content)`: incapsula la logica di creazione della componente grafica di visualizzazione.

2.3 createAboutControls(final JDialog window)

Il metodo ha il compito di generare la sezione della finestra "About" dotata del pulsante "Ok" di chiusura. Come parametro della funzione si ha l'oggetto JDialog `window`; leggendo il l'intero codice notiamo come questo sia in realtà l'oggetto `aboutBox` definito precedentemente: possiamo quindi rimuovere il parametro ed utilizzare direttamente l'oggetto. Inoltre è preferibile per il metodo un nome come `createAboutControlPanel()` per rendere chiaro che cosa si intende con "controls". Il metodo inoltre si occupa del sistema di "chiusura" nel momento in cui l'utente preme il pulsante "ok": anche in questo caso vogliamo dividere la logica dall'interfaccia grafica, estraendo il nuovo metodo `closeAboutDialog()`: sebbene sia possibile parametrizzare il metodo per specificare quale finestra di dialogo chiudere, si è scelto di mantenere il metodo senza argomenti data l'incertezza sul riutilizzo del metodo in altri contesti del progetto.

2.4 createAboutBox()

Il metodo `createAboutBox()` è responsabile dell'istanziazione e della configurazione iniziale della `JDialog`, definendo il titolo e la gerarchia dei componenti tramite il riferimento al frame principale. La leggibilità del metodo è stata migliorata rinominando la variabile `topContainer` in `parent`, e rimuovendo il ramo `else` superfluo, grazie alla presenza del `return` nel primo blocco condizionale.

2.5 Commenti e formattazione

La classe è priva di commenti interni, ad eccezione di un blocco esteso all'inizio del file che verrà rimosso; non è necessario aggiungere commenti poichè il codice è sufficientemente chiaro. L'ordinamento è stato riorganizzato seguendo la Stepdown Rule: in cima si hanno i metodi di alto livello, ovvero il costruttore e `actionPerformed`, seguiti immediatamente dai metodi che ne implementano i dettagli di medio livello

`initializeAboutBox`, `createAboutBox` e `setupLayout`. Scendendo ulteriormente, si incontrano i metodi puramente di implementazione a bassa astrazione, come `closeAboutDialog` e `buildCopyrightText`.

3. DefaultOverviewMouseMode

La classe funge da controller per l'interazione dell'utente con una finestra di "overview" della mappa. In particolare intercetta l'evento `mouseReleased` e decide come aggiornare la mappa in base all'azione dell'utente. In particolare, verrà eseguito il refactoring del metodo principale `mouseReleased(MouseEvent e)`.

3.1 mouseReleased(MouseEvent e)

Il metodo appare d'impatto confuso a causa dei numerosi calcoli matematici e le condizioni innestate. Si può immediatamente notare come è stato violato il principio della *Separation of Concerns*. È necessario estrarre da esso i diversi metodi ed utilizzare mouseReleased come orchestratore. Distinguiamo i metodi in base al principio CSQ; `recenterMap`, `performZoom` e `logDebugInfo` hanno il compito modificare la proiezione della mappa e stampare log (non restituendo dati), mentre `calculateLongitudeDifference`, `calculateDeltaDegrees` `calculateNewScale` prendono input e restituiscono un risultato (senza cambiare lo stato del programma).

Estraiamo quindi i seguenti metodi:

- `mouseReleased(MouseEvent event)`: il metodo fungerà da orchestratore per gli altri metodi estratti. Si limita a definire nel corretto ordine il flusso delle operazioni separate in debug, verifica, sincronizzazione e pulizia.
- `isInvalidInteraction(MouseEvent event)`: il metodo migliora la lunga formula logica originale `if (!(obj == theMap) || !autoZoom || point1 == null)` e quella del sua guardia precedente. Assegnando una variabile booleana a queste espressioni, le condizioni di validità diventano semanticamente chiare. La variabile `point1` non può essere rinominata poichè viene ereditata dalla classe `NavMouseMode2`, cosi come `theMap, autoZoom`.
- `processMouseRelease(MouseEvent event)`: lo scopo è incapsulare la logica dell'interazione utente, ovvero la registrazione dell'evento "click" e distinguerlo da un drag del mouse. Estraendo questa logica condizionale è stata eliminata la necessità di blocchi if/else annidati, poichè anche i corpi per ogni casistica sono stati separati in metodi opportuni.
- `isClickOrSmallDrag(int dx, int dy)`: il metodo funge da condizione nel blocco if di `processMouseRelease`. Nel codice originale, il valore 5 è ripetuto 4 volte per indicare la soglia di sensibilità del movimento del mouse: possiamo definirlo quindi in una costante `DRAG_THRESHOLD`.

- `recenterMap(Point2D clickPoint)`: vogliamo isolare l'azione del ricentramento della mappa dalla logica geometrica che calcola le coordinate, nel caso in cui l'utente esegua un click.
- `performZoom(int deltaX, int deltaY)`: eseguiamo il calcolo matematico e l'aggiornamento dell'interfaccia nel caso in cui l'utente esegua un azione di drag sulla mappa per zoommare. Delega il calcolo del fattore di scala al metodo `calculateNewScale`, che si occupa esclusivamente di applicare i risultati di scala e nuovo centro.
- `calculateLongitudeDifference`, `calculateDeltaDegrees`, `calculateNewScale`: i 3 metodi con nome autoesplicativo restituiscono il risultato del calcolo delle misure richieste dai metodi menzionati precedentemente.

3.2 Commenti e formattazione

Nella classe sono presenti numerosi commenti all'interno del metodo mouseReleased (come `// recenter the map, // allow for crossing dateline // Figure out the new scale`) per spiegare blocchi di codice complessi e poco chiari. Questi commenti rappresentano un fallimento nell'esprimersi attraverso il codice e generano inutile "rumore" visivo. Dopo il refactoring, questi commenti devono essere rimossi poiché la loro funzione viene completamente assolta dai nuovi metodi e variabili con nomi autoesplicativi.

Applicando la Stepdown Rule, il punto di ingresso è `mouseReleased`, che definisce l'alto livello della gestione dell'evento. Successivamente, si incontrano i metodi chiamati in esso, come `isInvalidInteraction` e `processMouseRelease`, procedendo poi con un effetto a cascata dove vengono posizionati in ordine i metodi chiamati negli altri menzionati.

4. ImageServerUtils

ImageServerUtils è una classe di utilità con funzioni statiche progettate per facilitare l'elaborazione delle richieste di immagini di mappe. Il suo scopo è convertire dati di configurazione in oggetti di dominio utilizzati da Open Map.

4.1 createOMProjection(Properties props, Projection defaultProj)

Il metodo ha la responsabilità di istanziare e configurare l'oggetto di proiezione della mappa, estraendo i parametri essenziali dalle proprietà fornite. Esso determina la classe specifica di proiezione da utilizzare e coordina l'invocazione della `ProjectionFactory` per la generazione dell'oggetto finale. Il metodo viola il principio SRP in quanto si concentra su molteplici compiti: è necessario isolare le diverse funzionalità in metodi distinti. L'oggetto `Proj`, restituito dal metodo è composto da tipo di proiezione, coordinate del centro, scala, larghezza e altezza:

- `calculateCenterPoint` isola l'estrazione delle coordinate, rinominando la variabile ambigua `llp` in `defaultCenter`
- `getProjectionClassFromProperties` incapsula la logica decisionale per la selezione del tipo di proiezione Java corretta, nascondendo i dettagli di basso livello

E' stato deciso di non separare il calcolo delle misure di scala, larghezza e altezza poichè sono molto simili e leggibili ed estrarle in un metodo dedicato introduce una frammentazione eccessiva. Viene introdotto anche il metodo `logProjectionDebug` contenente tutti i log di debug, rimuovendo il "rumore" visivo dal metodo principale, dove rimarranno gli assegnamenti per l'assegnazione diretta delle 3 proprietà rimanenti a partire dal parametro `props`.

4.2 getBackground(Properties props, Paint defPaint)

Il metodo ha lo scopo di creare un colore a partire dalle proprietà e nel caso in cui queste non sono definite, usarne uno di default. Il metodo mescola la logica di estrazione dei dati, la manipolazione della trasparenza e la gestione del logging; per questo è necessario separare la ricerca del colore base nel metodo `determineBasePaint` e

la logica di modifica della trasparenza in `applyTransparency`. Il log viene incapsulato in `logBackgroundDetails`, riducendo il rumore nel codice. Il problema che ora emerge riguarda la gestione dei due parametri ridondanti `props` e `defPaint` nei metodi `getBackground` e `determineBasePaint`, cosi come per `applyTransparency` e `logBackgroundDetails`. Una possibile soluzione è trasformare questi parametri in campi della classe o raggruppandoli in un oggetto di configurazione dedicato; questa modifica però è strettamente legata alla natura dei metodi e dalle scelte dello sviluppatore del codice, motivo per cui il codice non verrà modificato sotto quest'aspetto.

4.3 Commenti e formattazione

La classe presenta commenti Javadoc ridondanti che vogliono spiegare il ruolo dei parametri per ogni metodo, nonostante sia già abbastanza chiaro dai metodi stessi. Anche in questo caso, il refactoring apportato permette di rimuovere la maggior parte di questi commenti in quanto ormai obsoleti e solo una fonte di rumore. Applichiamo ancora una volta la Stepdown Rule seguendo i livelli di astrazione dei metodi, a partire da `createOMProjection` fino al metodo `logBackgroundDetails`.

5. ESRIRecord

La classe *entity* `ESRIRecord` è il modello astratto utilizzato per leggere e gestire le informazioni contenute in un file cartografico. Il suo scopo principale è quindi quello di incapsulare i dati e la struttura dell'intestazione di ogni record geografico.

I principi di Clean Code, data la natura della classe, verranno adottati per affinare la definizione di parametri e proprietà, migliorare la qualità dei commenti e ottimizzare la leggibilità generale.

5.1 Miglioramento della leggibilità

Si riscontra un'abbondanza di commenti ridondanti che si limitano a ripetere ciò che già il nome del metodo o del parametro comunicano chiaramente, appesantendo la lettura del codice senza aggiungere valore; è presente anche un commento che descrive vincoli di compatibilità legati a specifiche versioni del JDK, informazione che dovrebbe essere riportata non nel codice ma nella documentazione ufficiale. Viene infine riportata una parte di codice non utilizzato come commento. Vogliamo quindi rimuovere questi blocchi al fine di rendere la lettura del codice più veloce e chiara.

5.2 Utilizzo di nomi significativi

L'uso di commenti per spiegare variabili dai nomi poco chiari, come `b` per indicare un buffer, è un segno di nomenclatura poco chiara. Questa criticità si risolve applicando il principio dei nomi significativi: rinominando le variabili in modo che il loro scopo sia evidente e sfruttando la chiarezza del tipo di dato (in particolare nei parametri dei metodi), il codice diventa capace di spiegarsi da solo: ad esempio `byte[] buffer`. In questo modo, i commenti diventano superflui e possono essere rimossi, ripulendo il codice.

5.3 Costanti e incapsulamento

L'uso di costanti al posto dei valori numerici elimina l'ambiguità semantica poichè esplicita a cosa è riferito tale numero nel momento in cui una persona legge il codice; questa regola deve essere applicata alle varie dimensioni usate per la creazione del buffer. Definiamo quindi le costanti `RECORD_HEADER_SIZE_BYTES`, `CONTENT_LENGTH_OFFSET_BYTES`, e `WORD_SIZE_BYTES`.

All'inizio della classe troviamo le variabili `recordNumber` e `contentLength`, dichiarate come pubbliche. Tuttavia, la presenza di appositi metodi get() suggerisce che l'accesso a questi dati debba essere controllato. Per seguire correttamente il principio dell'incapsulamento, queste variabili andrebbero dichiarate come private, proteggendole quindi da modifiche indesiderate.

6. HelloWorldLayer

La classe HelloWorldLayer di esempio che serve a dimostrare il funzionamento di oggetti geometrici su una mappa OpenMap. Il suo scopo è rappresentare la scritta "HELLO WORLD" utilizzando l'oggetto poligono definito nella classe `OMPoly`.

6.1 createGraphics(OMGraphicList list)

Originariamente, le coordinate della scritta sono definite come `double[]` direttamente nel corpo del metodo, insieme alla logica di creazione dei poligoni. Questa struttura viola il principio di separazione tra dati e logica. Il metodo risultava eccessivamente lungo e ridondante, poiché replicava la stessa istanziazione per ogni elemento grafico, Per rispettare SRP, i dati sono stati estratti in un file di esterno, delegando al metodo il compito di orchestrare la generazione grafica. Creiamo quindi il file `coordinates.properties` contenente tutte le cooordinate ed il metodo `initializeLayerGraphics` che, grazie ai metodi ausiliari `loadProperties` per il caricamento del file e `parseCoordinates`, applica il metodo `addPolygonToLayer` ad ogni coordinata.

6.2 Commenti e formattazione

I tre metodi `paint`, `projectionChanged` e `setProperties` presentano commenti che si limitano a esplicitare il ruolo dei parametri ricevuti quando è già chiaro dal contesto dei metodi stessi. Tali annotazioni sono considerate ridondanti e possono essere rimosse. Per quanto riguarda l'ordinamento dei metodi, si trovano prima le costanti e i costruttori che definiscono lo stato iniziale dell'oggetto e poi immediatamente i metodi astratti, che rappresentano il livello di astrazione più alto. Successivamente, si trovano i metodi che implementano la logica di scrittura e le funzioni di set/get.