Testo Esercizio

Si consideri la realizzazione di un semplice programma grafico per il disegno di figure geometriche in due dimensioni. Si analizzino i requisiti e se ne rappresentino i risultati in UML identificando, in particolare, le principali gerarchie.
Si rappresenti mediante un diagramma di sequenza lo scenario di creazione e colorazione di una figura geometrica (a scelta).

Sommario

- Parte 1(vista esterna, aspetti use case driven) – rappresentazione dei requisiti in UML;
- Parte 2 (vista logica, aspetti strutturali) – rappresentazione delle gerarchie e delle classi principali del sistema in UML;
- Parte 3 (vista logica, aspetti dinamici) – rappresentazione di uno scenario d'utilizzo del sistema mediante diagrammi di sequenza.

Note.
La soluzione proposta è solo una possibile soluzione. In generale, non esiste la "soluzione corretta" o il "modello migliore". I modelli si costruiscono esattamente come avviene per qualsiasi artefatto umano. Molti dettagli possono essere soggettivi e dipendono fondamentalmente dalla capacità di astrazione, dallo spirito di osservazione e dalla creatività dell'autore. Questa proposta di soluzione rappresenta solo una linea guida: essa costituisce un invito a sperimentare, ad essere creativi, ma allo stesso tempo ad essere coerenti. Gli aspetti fondamentali di un modello UML sono i seguenti:

- Individuare i concetti essenziali del sistema e rappresentarli opportunamente;
- Caratterizzare ciascun concetto con il livello di dettaglio utile per risolvere il problema;
- Mantenere la coerenza tra le diverse viste (o prospettive) del modello. Se mostriamo un'interazione tra due oggetti in un diagramma di sequenza, per esempio, il metodo descritto nel diagramma deve essere stato prima definito nella classe, con gli opportuni argomenti (se ce ne sono).

Un modello è ragionevole quando contiene queste tre caratteristiche.

Svolgimento

Parte 1

Il requisito più ovvio di un programma di disegno è sicuramente quello di consentire all'utente di disegnare delle figure geometriche sullo schermo. Identifichiamo, quindi, un caso d'uso generale che chiamiamo "CreaFigura". Casi particolari di tale caso d'uso sono quelli per la creazione di un cerchio, di una linea, di un rettangolo, e così via. Esprimiamo questa relazione mediante la generalizzazione tra i casi d'uso, come illustrato in Figura 1.

Assumiamo, inoltre, che lo scenario di creazione di una figura sia completo solamente quando tale figura viene disegnata sullo schermo (solo dopo questo momento, infatti, l'utente percepisce il valore del caso d'uso, ossia un suo effetto). Aggiungiamo quindi un caso d'uso "DisegnaFigura" che è collegato al caso d'uso generale "CreaFigura" mediante una relazione «include».

Un altro ovvio caso d'uso è quello che permette ad un utente di colorare una figura. Quest'attività può essere scomposta (ancora relazione «include») nelle attività di

- impostazione di un colore come colore corrente (o colore attivo);
- (ri)disegno della figura stessa.

Altri requisiti tipici di un programma di disegno sono quelli che permettono di spostare una figura, di cancellarla, di cambiarne le dimensioni, di stampare il disegno, di creare un nuovo disegno, di salvare il disegno corrente. La figura 1 illustra alcuni di questi requisiti e le relazioni tra i casi d'uso rappresentati.

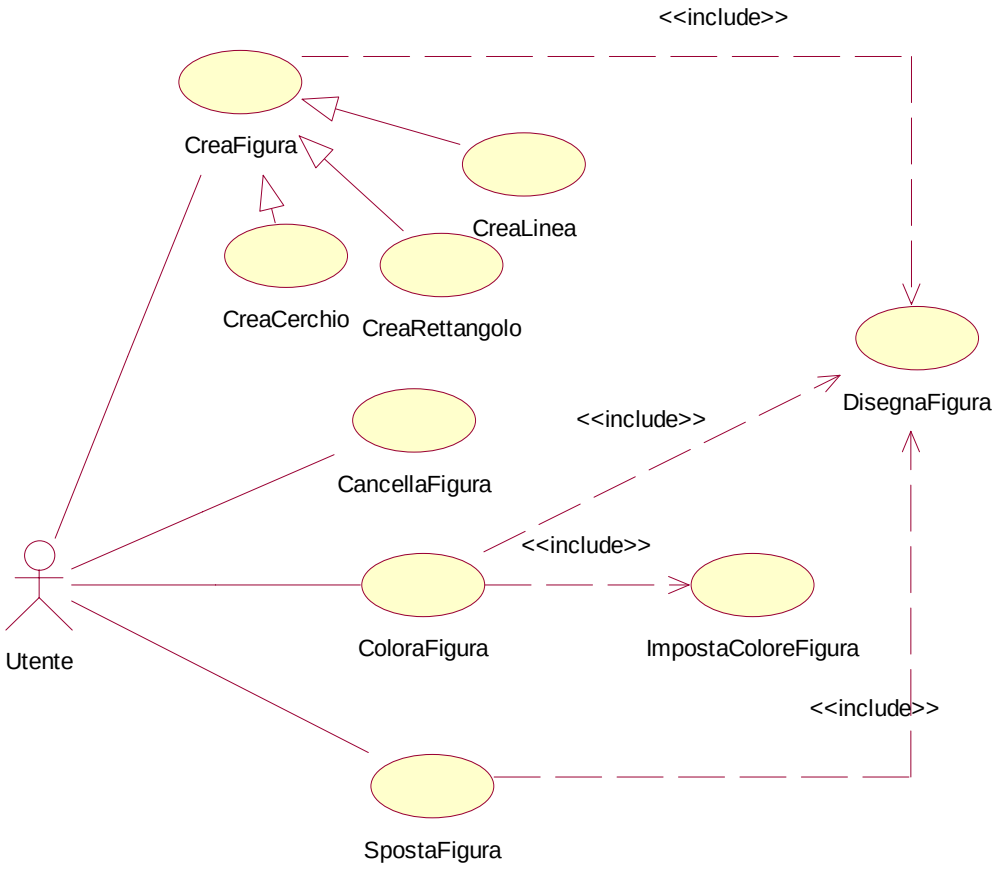

Figura 1 - Diagramma di alcuni casi d'uso per l'esercizio proposto

Parte 2

Il primo passo per costruire dei diagrammi di classe consiste nell'identificare le principali classi che costituiscono il sistema. Una tecnica molto utile per iniziare è quella di effettuare un'analisi linguistica del tema proposto. Il testo dell'esercizio descrive le funzionalità generali del sistema e il contesto d'utilizzo. Nell'esempio proposto, dobbiamo caratterizzare un programma di grafica che permette di disegnare delle figure geometriche presumibilmente su una finestra di disegno che fa parte del programma stesso. L'utente, inoltre, deve poter compiere delle operazioni sulle figure, quali (ad esempio) la colorazione (un chiaro richiamo ai requisiti e ai casi d'uso: le diverse viste di un modello si complimentano!).

Iniziamo quindi con l'identificazione delle principali gerarchie. La prima gerarchia ovvia è quella delle figure. Immaginiamo di poter disegnare le più semplici primitive grafiche, quali: punti, linee, rettangoli, quadrati, cerchi, triangoli (Figura 2).

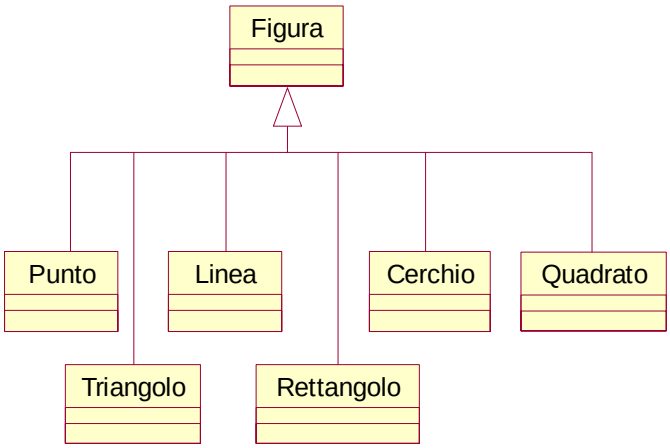

Figura 2 - Gerarchia Figura

Dopo aver individuato le classi di una gerarchia, dobbiamo passare a caratterizzarne alcuni *essenziali* dettagli interni. In particolare, è necessario identificare i principali attributi e metodi che caratterizzano le classi. Solitamente si inizia dalla classe base. Il concetto di figura è un concetto astratto, utile per raggruppare *proprietà comuni* a tutti i tipi di figura. In generale, quindi, possiamo pensare che *ogni* figura abbia (ad esempio) un baricentro, espresso mediante le sue coordinate x e y, e un colore, espresso mediante le componenti RGB (ogni altra rappresentazione andrebbe ugualmente bene). Ogni figura, inoltre, può avere un'area (per ipotesi, assumiamo nulla l'area delle figure che non hanno una parte interna, come il punto e la linea). Abbiamo individuato quindi i seguenti attributi:

- *xBaricentro* (di tipo intero);
- *yBaricentro* (di tipo intero);
- *area* (di tipo double);
- *colore* (che possiamo rappresentare con una classe aventi come attributi le componenti intere RGB *rComp*, *gComp* e *bComp*).

Caratterizziamo ora la classe *Figura* con dei metodi. Ipotizziamo per semplicità che essa abbia per ogni attributo i corrispondenti metodi accessori *get()* e *set()*, non mostrati nei successivi diagrammi di classe solo per convenzione tipografica (riduciamo il livello di dettaglio ai soli aspetti essenziali: basta vedere gli attributi per intuire la presenza di tali metodi: aggiungerli esplicitamente non aumenterebbe l'informazione complessiva). Poiché ogni figura deve essere disegnata, è evidente la necessità di un metodo di disegno che chiamiamo *disegna()*:

- *void disegna();* // disegna una generica figura

Tale metodo sarà probabilmente vuoto nella classe *Figura* (non sappiamo disegnare una generica figura!), ma sfruttando l'ereditarietà e il polimorfismo, possiamo ridefinirlo per ogni figura geometrica specifica (ogni particolare figura sa come disegnarsi!).
Altri metodi ovvi, descritti in Figura 3, sono i seguenti:

- *void impostaColore(Colore colore);* // imposta il colore corrente, ma non lo applica ancora
- *void riempiColore(Colore colore);* // applica il colore alla parte interna della figura

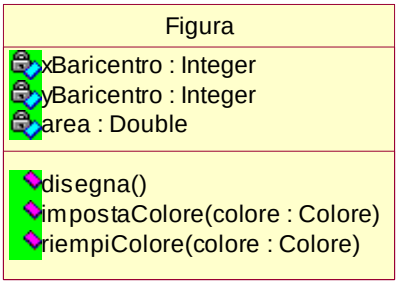

Figura 3 - La classe base Figura con i suoi attributi e i suoi metodi

Definita la classe base, caratterizziamo alcune classi derivate. Ora dobbiamo analizzare singolarmente ciascuna particolare figura geometrica. Un punto, ad esempio, avrà sicuramente delle coordinate x e y. Potremmo osservare, in questo caso, che le coordinate che descrivono la sua posizione sono uguali a quelle che descrivono il suo baricentro. Tuttavia baricentro e posizione sono due concetti (in generale) differenti. Nel modello vogliamo mantenere separate queste due caratteristiche (il baricentro di un rettangolo può essere diverso dalla posizione: potrei voler posizionare un rettangolo sfruttando uno dei suoi vertici, piuttosto che il cento che è un punto non immediatamente percepibile con precisione da un osservatore). Una linea, invece, può essere descritta mediante due punti: un punto di origine e un punto di destinazione.

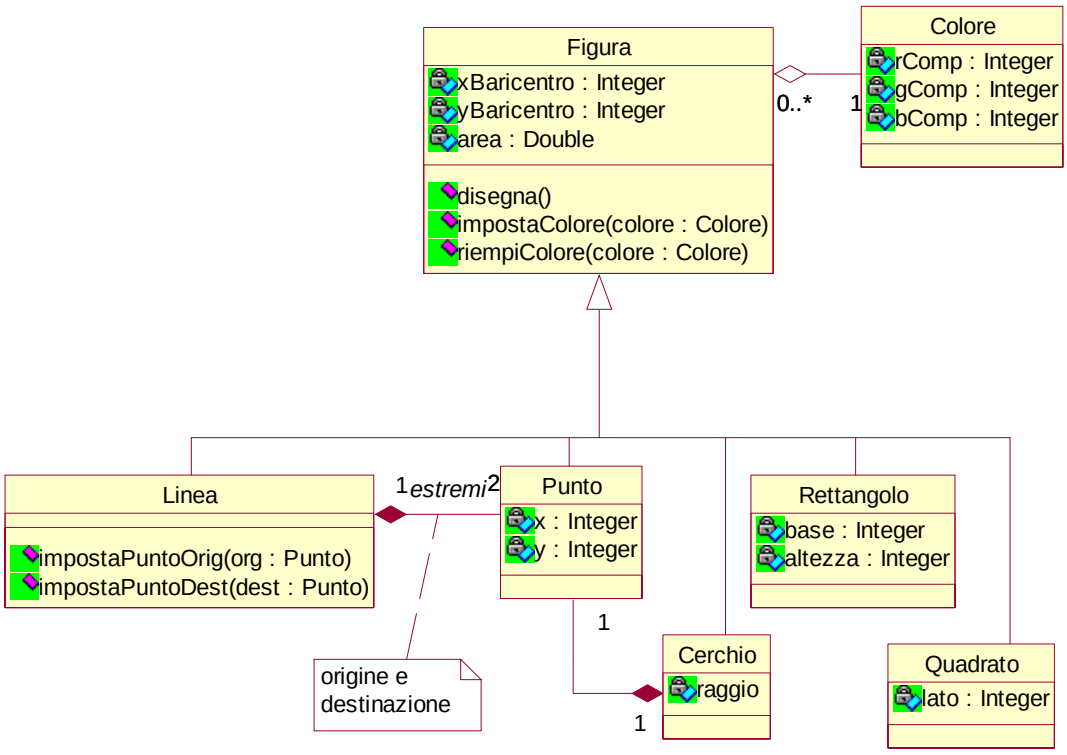

Figura 4 - La gerarchia Figura

Possiamo riutilizzare, quindi, la classe *Punto* per modellare la classe *Linea*, utilizzando la composizione (una linea è strutturalmente composta da due punti che chiamiamo estremi). Tale classe potrebbe avere due metodi che impostano tali punti. Il cerchio, a sua volta, può essere caratterizzato da un *raggio* e da un *centro* (anche quest'ultimo è un punto). Mettiamo ora tutto nel diagramma UML della gerarchia *Figura*, riportato in Figura 4.

Identifichiamo ora altre gerarchie. Se pensiamo ad un programma di grafica, immaginiamo subito che esso abbia una GUI costituita da alcuni menu di comandi. Abbiamo quindi individuato due concetti: menu e comandi. Molti programmi di grafica, inoltre, sono dotati di una palette delle figure che contiene un'icona che rappresenta ogni particolare figura geometrica che possiamo disegnare (per mezzo di una semplice operazione di drag&drop). Esempi di comandi sono tutti quelli relativi ai file (Crea, Apri, Salva, Salva con Nome,…), oppure i file di Edit (Copia, Incolla, Taglia,…). Possiamo pensare a questo punto alla GUI come a un oggetto che aggrega un menu di comandi e una palette delle figure. La Figura 5 illustra la gerarchia dei comandi[1], mentre la Figura 6 illustra l'oggetto *GUI*.

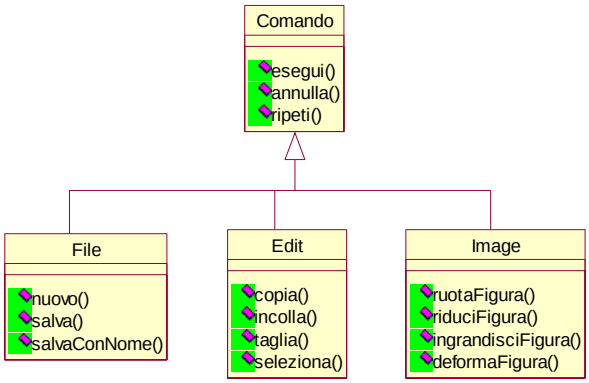

Figura 5 – La gerarchia dei comandi

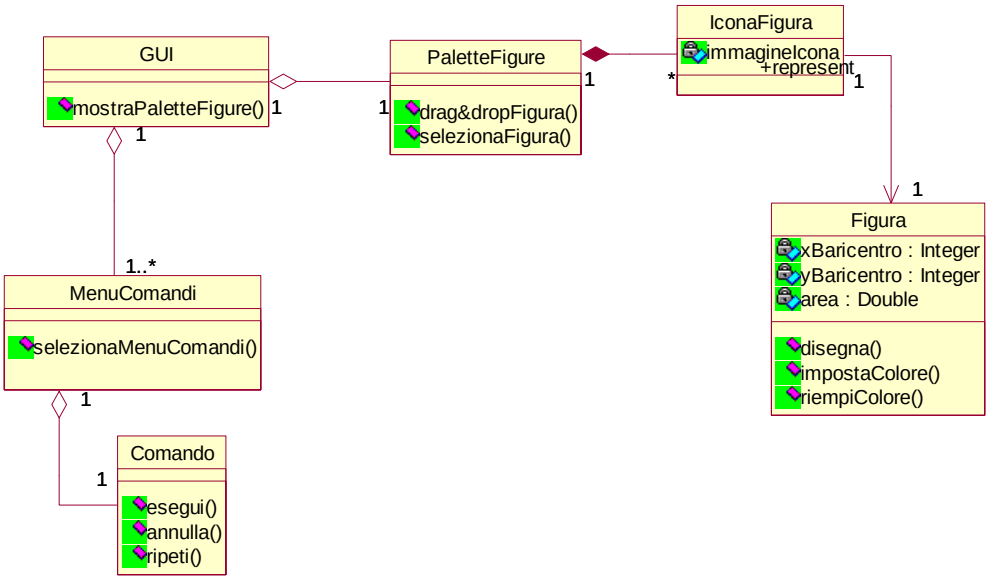

Figura 6 - La GUI costituita da un menu comandi e da una palette delle figure.

[1] Una soluzione alternativa potrebbe essere quella di identificare il concetto generale di menu. Particolari tipi di menu potrebbero essere il menu dei file, il menu di Edit, il menu Image, eccetera. Ciascuno di questi menu potrebbe aggregare diversi tipi di comando: i comandi *nuovoFile*, *Salva*, *SalvaConNome* per quanto riguarda i file, e così via per gli altri menu. Avremmo due gerarchie in relazione tra loro (per ragioni di semplicità questa soluzione non viene qui mostrata).

Arrivati a questo punto abbiamo identificato figure, comandi, GUI, menu e palette. Per completare lo scenario di disegno proposto abbiamo bisogno di un ulteriore oggetto: la finestra di disegno. La finestra di disegno deve consentire all'utente di collocare una figura sullo schermo, di disegnarla, di colorarla e, in generale, di effettuare tutte le operazioni sulle figure (attraverso il menu dei comando oppure la palette delle figure).

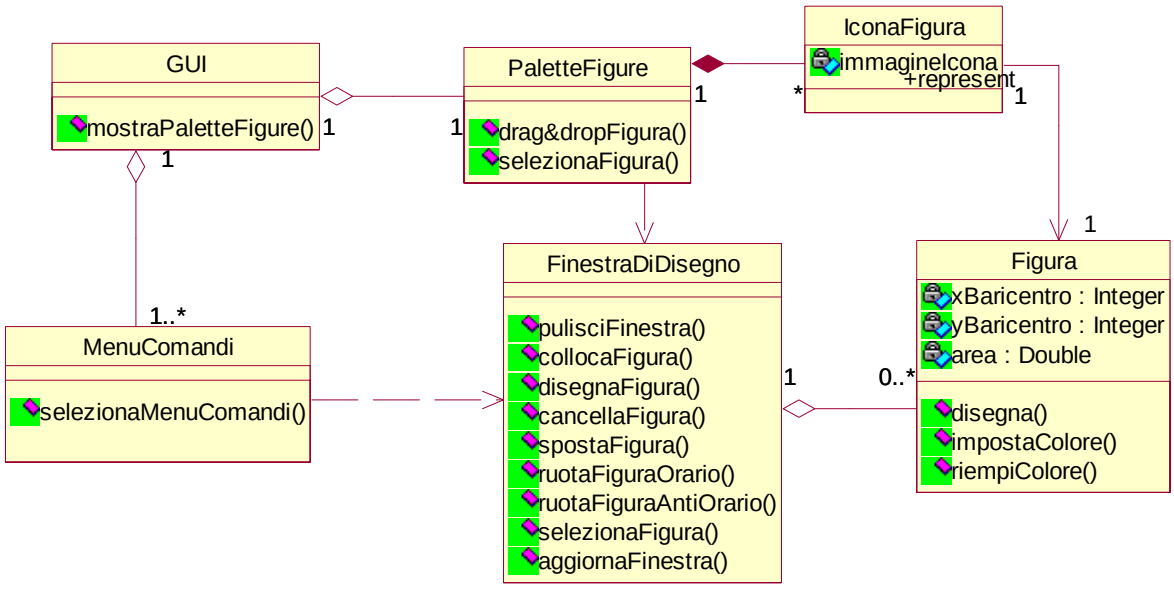

Figura 7 - La Finestra di disegno

Aggiungiamo infine una classe utente e mettiamo assieme tutti i concetti, omettendo tutti i dettagli relativi alle segnature delle operazioni (Figura 8).

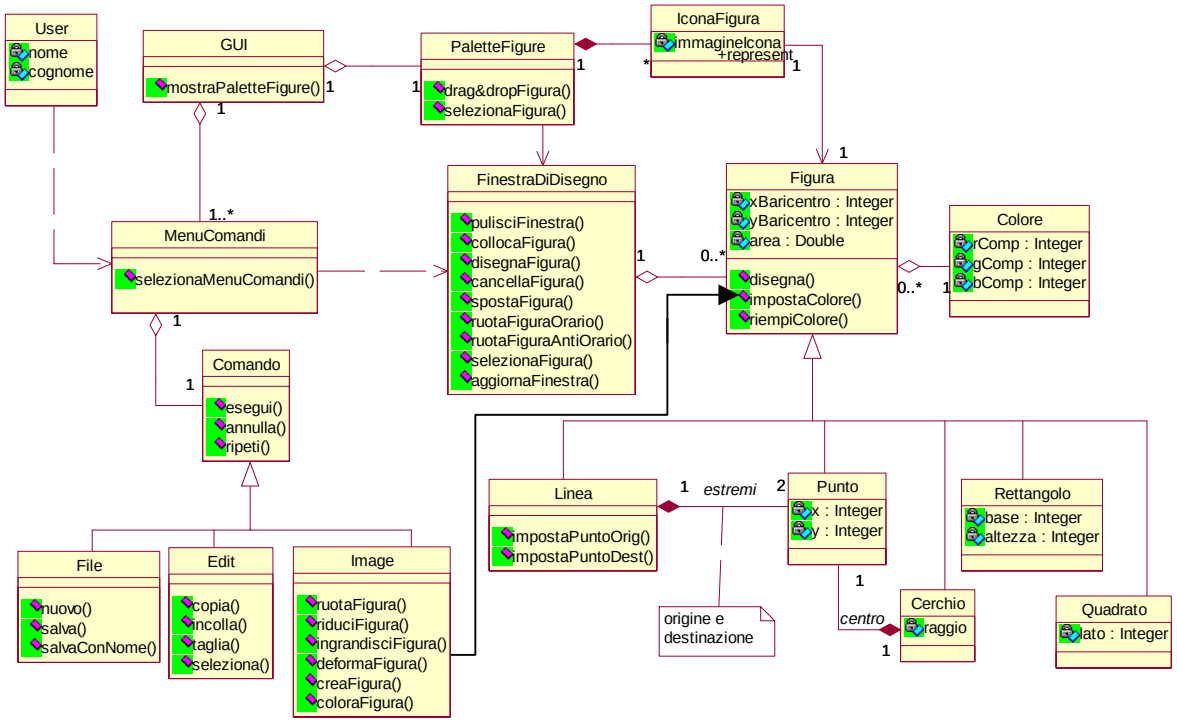

Figura 8 - Tutti gli elementi della soluzione proposta in un diagramma di classe

Parte 3

Realizziamo ora lo scenario richiesto dall'esercizio. Un utente seleziona il menu dei comandi relativi alla gestione delle immagini (menu Image), quindi crea una nuova figura (ad esempio un rettangolo). La creazione del rettangolo viene realizzata dal metodo esegui() del comando Image.creaFigura(). Questo metodo, a sua volta, deve creare un nuovo oggetto di tipo Rettangolo, lo colloca nell'angolo in alto a sinistra della finestra di disegno (metodo FinestraDiDisegno.collocaFigura()) ed, infine, deve disegnarlo sullo schermo. L'operazione di disegno viene eseguita inizialmente dalla finestra di disegno, attraverso il metodo disegnaFigura. La finestra di disegno, a sua volta, chiama il metodo disegna() della classe Rettangolo che contiene il codice per disegnare un rettangolo.
Lo scenario di utilizzo richiesto è completato dal riempimento della parte interna di una figura con un colore scelto dall'utente. L'utente interagisce con il menu Image, specificando la figura su cui applicare il colore e il colore stesso. Image.coloraFigura() deve prima impostare il colore scelto dall'utente come colore corrente e poi colorare la figura. L'interazione con la finestra di disegno e con la classe Rettangolo è analoga allo scenario precedente. La Figura 8 illustra questi due scenari di utilizzo del sistema mediante il diagramma di sequenza richiesto.

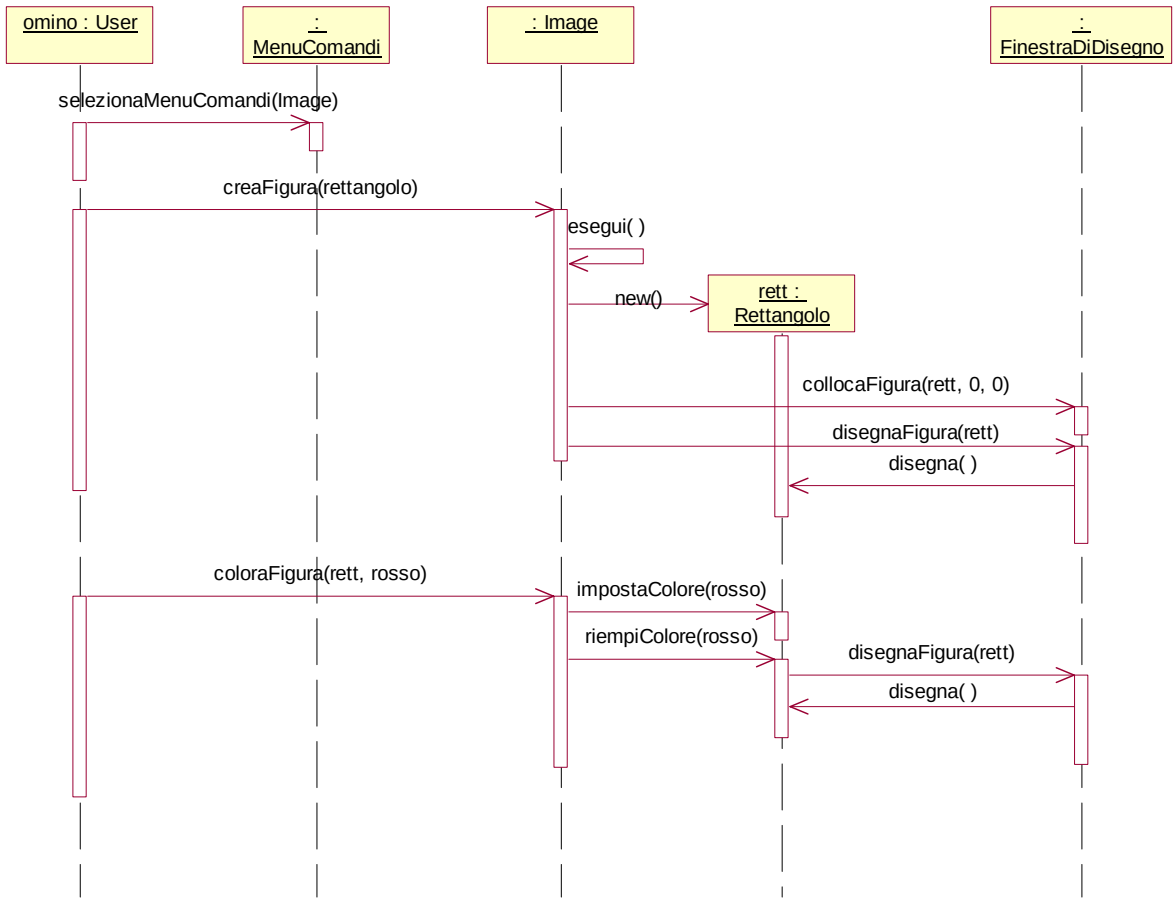

Figura 9 – Creazione di una nuova figura e sua colorazione.