

# Fondamenti di Ingegneria del Software a.a. 2024/2025

## Laboratorio 3 – UML Class & Sequence Diagrams

### Si consideri la descrizione parziale del seguente sistema RubricaSYS:

*“L’utente utilizza RubricaSYS per gestire la rubrica telefonica personale, che include una lista di contatti (massimo 100). Ogni contatto è caratterizzato da un nome univoco, una descrizione, un valore booleano che stabilisce se il contatto è protetto o meno (impedendone la cancellazione nel primo caso), e un numero di telefono. L’utente può inserire nuovi contatti nella rubrica telefonica, rimuovere quelli presenti, o aggiornarne i dati [...]”*

### Si richiede di:

1. Replicare (da pagina 2 di questo documento) la modellazione guidata per l’operazione **“inserisciNuovoContatto”**, descritta come di seguito:

*“L’utente richiede di inserire un nuovo contatto in rubrica. Il sistema crea un contatto inizialmente vuoto e lo aggiunge alla rubrica dell’utente”*

2. Modellare in modo autonomo l’operazione **“inserisciDatiContatto”**, descritta come di seguito, producendo un nuovo diagramma di sequenza e aggiornando in modo coerente il diagramma delle classi prodotto al punto precedente:

*“L’utente inserisce i dettagli per il contatto creato precedentemente: nome, descrizione, protetto (vero, se il contatto è protetto da cancellazioni), e numero di telefono. Il sistema aggiorna il contatto nella rubrica dell’utente coi dati inseriti”*

### Modalità di consegna:

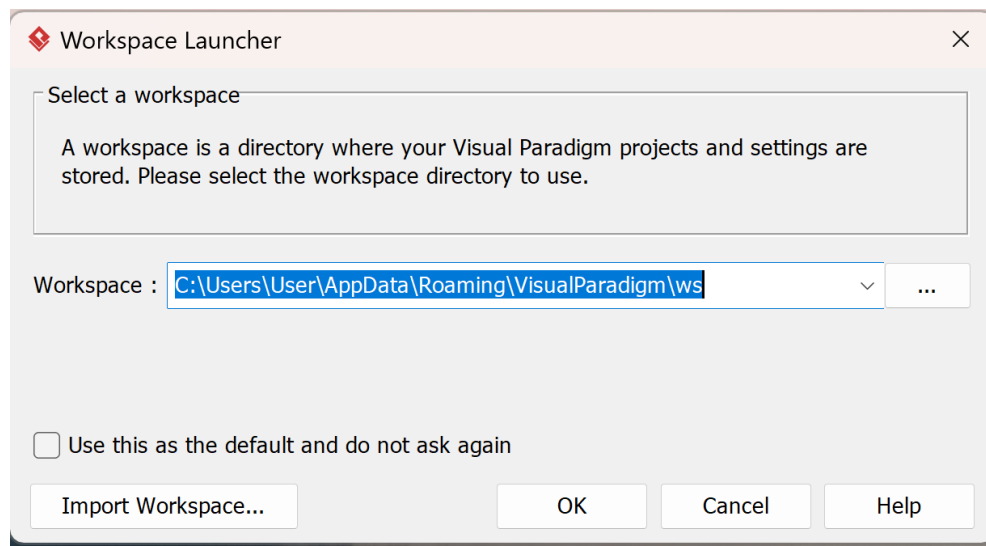
Caricare nella pagina del corso su Aulaweb uno ZIP contenente: (i) file README che includa nomi e matricole dei membri del gruppo, e, (ii) file .VPP relativo al modello UML prodotto in Visual Paradigm relativamente alle operazioni **inserisciNuovoContatto** e **inserisciDatiContatto**.

**Scadenza: 25/11/2024**

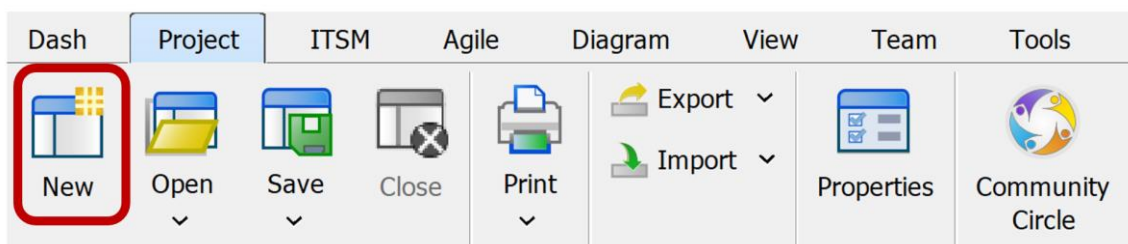
## Modellazione guidata per “inserisciNuovoContatto”

Per questo task, ci affideremo a **Visual Paradigm (VP)**, un tool per la modellazione e la progettazione di applicazioni software, che supporta tutti i modelli previsti dallo standard UML 2. L'edizione Community di VP (versione 17.2, Novembre 2024) è scaricabile al link <https://www.visual-paradigm.com/download/community.jsp>.

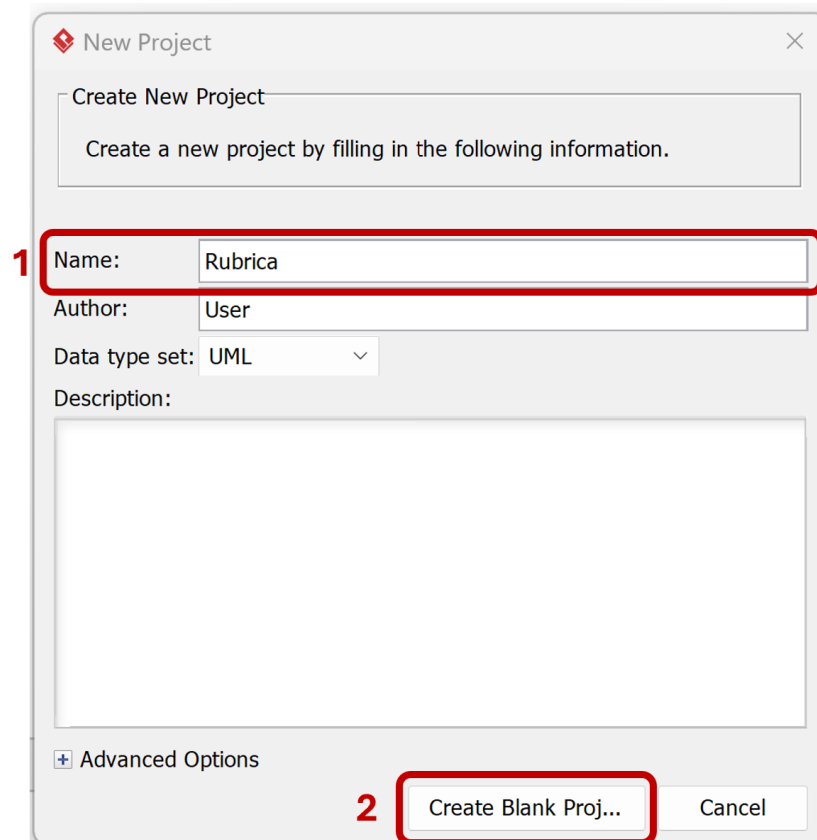
Una volta installato e avviato VP, come prima cosa verrà richiesto di indicare il path al workspace. Va bene lasciare il path di default, ma va bene anche un path che punta a una cartella dedicata (es. *C:\Users\User\Desktop\lab3*).



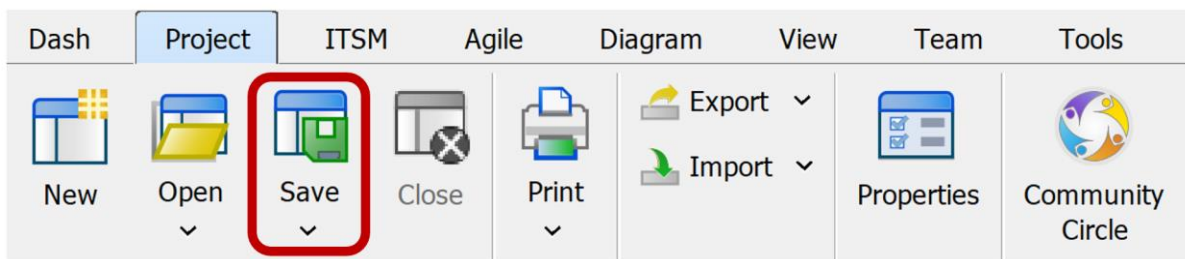
Al primo avvio, VP aprirà una serie di finestre di guida, che si possono chiudere. Come passo seguente, è necessario creare un progetto. Dal menu principale, selezionare **Project > New**.



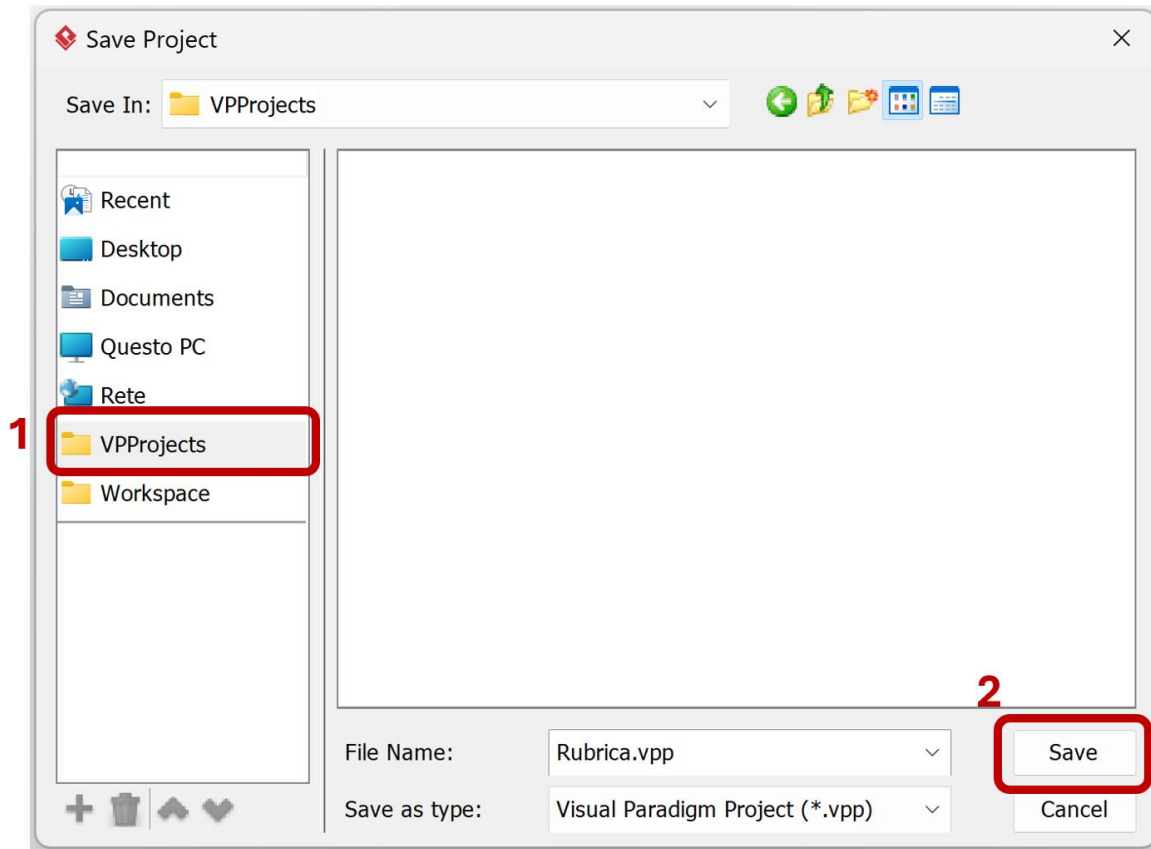
Nella finestra successiva, indicare alla voce **Name** il nome del progetto, che chiameremo “Rubrica”, e premere **Create Blank Project**.



A questo punto, il progetto (vuoto) è stato creato. Ora salviamo il file, cliccando su **Save** sotto la scheda **Project**.



Selezioniamo il percorso in cui salvare il file (formato .VPP), indicando la cartella di destinazione (es. *VPPProjects*, ma una qualsiasi cartella a scelta dentro il workspace va bene) e cliccando sul bottone **Save**.






Il file salvato verrà rappresentato nel modo seguente.

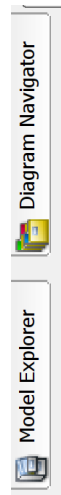


Rubrica

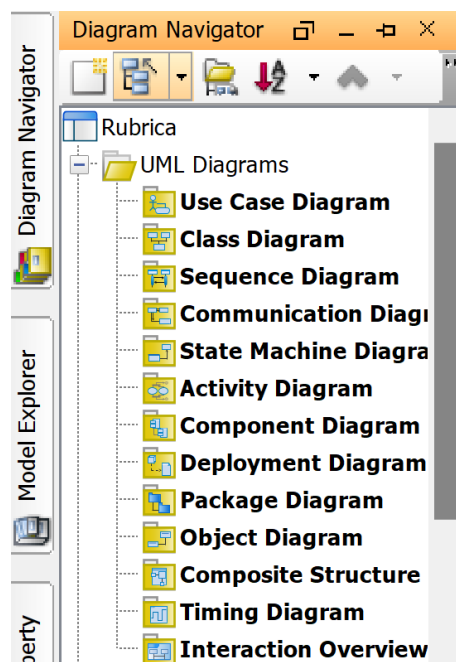
Nel corso della modellazione, nella cartella di salvataggio del progetto potreste notare alcuni file “misteriosi”. Si tratta principalmente di file relativi a lock e backup, autogenerati a seguito delle modifiche e salvataggi. Nel corso della modellazione, il numero dei file di backup potrebbe crescere molto, pertanto ogni tanto può aver senso cancellare i file più vecchi autogenerati.

 .Rubrica.vpp.lck	01/03/2024 15:45	File LCK	0 KB
 Rubrica	01/03/2024 15:45	Visual Paradigm P...	592 KB
 Rubrica.vpp.bak_000f	01/03/2024 15:45	File BAK_000F	536 KB

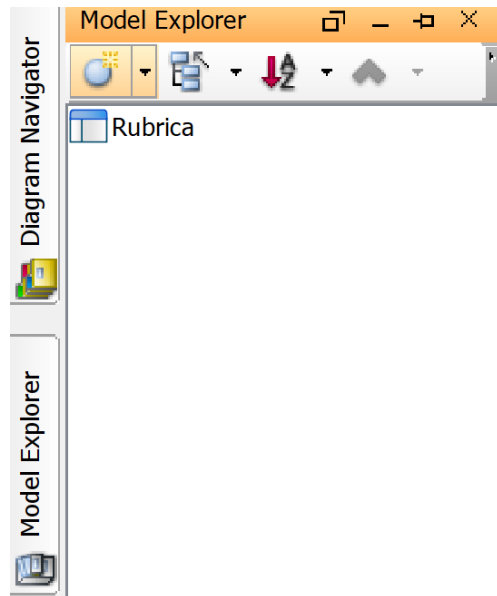
Ora che il progetto è salvato, fermiamoci a osservare alcuni elementi sull'estremità sinistra della schermata principale, in particolare le etichette **Diagram Navigator** e **Model Explorer**.



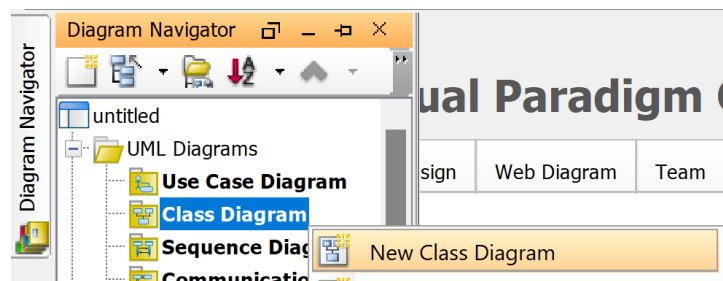
Cliccando col tasto sinistro del mouse su ognuna di queste voci, potremo espandere la scheda relativa. **La scheda Diagram Navigator elenca i diagrammi UML presenti nel progetto, ossia le viste, raggruppati per tipologia.** Per il momento l'elenco è ovviamente vuoto.



**La scheda Model Explorer invece rappresenta il modello UML**, raggruppando tutte le entità UML create (diagrammi, classi, package, attori, casi d'uso, lifeline, etc), preservando la consistenza, e permettendo di riusarle coerentemente durante la modellazione. Ad esempio, un concetto X può essere modellato staticamente attraverso una classe di un diagramma delle classi e dinamicamente mediante una lifeline di un diagramma di sequenza. Pertanto, come vedremo, si potranno collegare concetti come classi e lifeline per rappresentare sia aspetti statici che dinamici di un sistema, i quali verranno mantenuti aggiornati nel modello. Anche in questo caso, espandendo la scheda col click del mouse, per ora visualizzeremo un elenco vuoto, dal momento che dobbiamo ancora modellare.



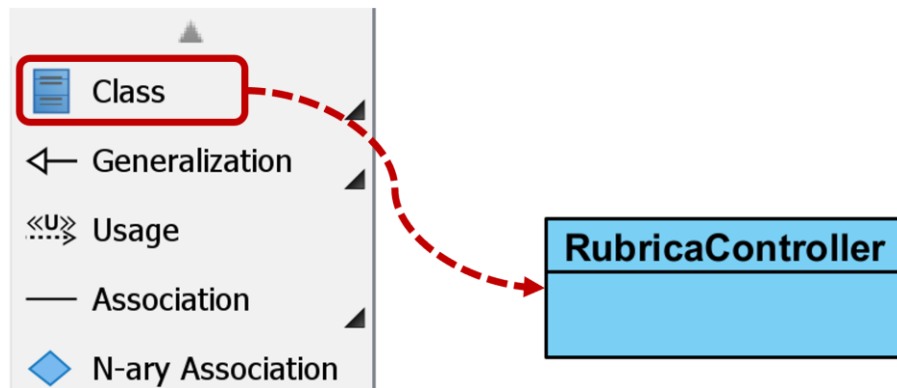
Mentre modelleremo l'operazione **“inserisciNuovoContatto”** in termini di diagramma di sequenza, in parallelo produrremo il corrispettivo diagramma delle classi, che rifletterà le entità coinvolte. Come punto di partenza, creiamo dunque un diagramma delle classi. Dal pannello **Diagram Navigator** a sinistra, una volta espansa la voce **UML Diagrams**, clicchiamo col tasto destro su **Class Diagram**, quindi selezioniamo la prima voce **New Class Diagram**.



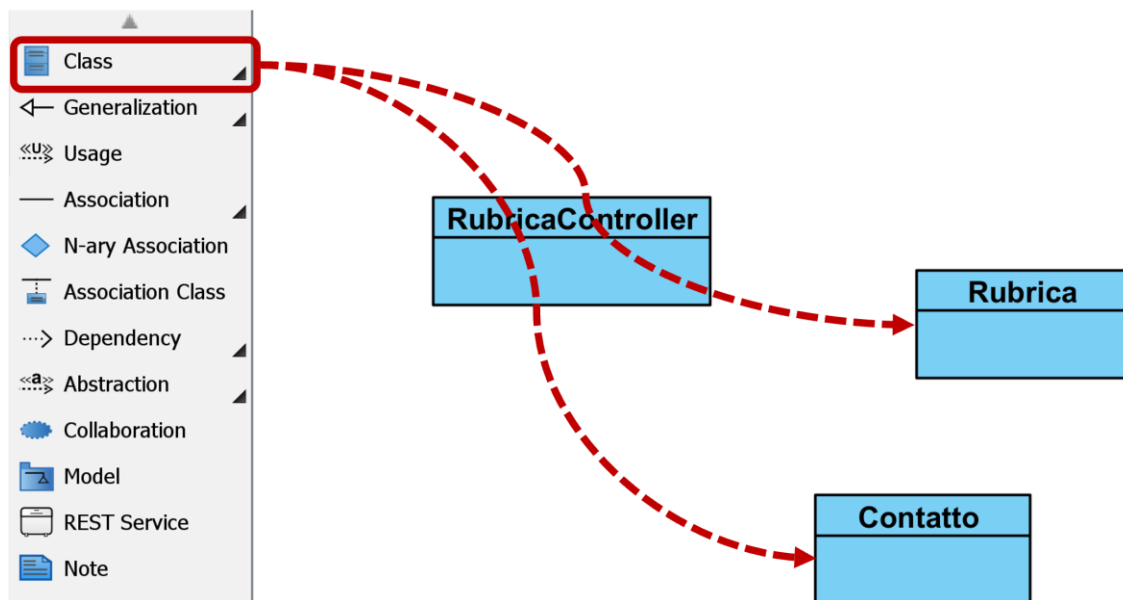
Verrà creato un diagramma delle classi vuoto. Facendo doppio click sul nome della scheda **“<default package>”** in alto nel diagramma, rinominiamola in **“Diagramma classi Rubrica”**.



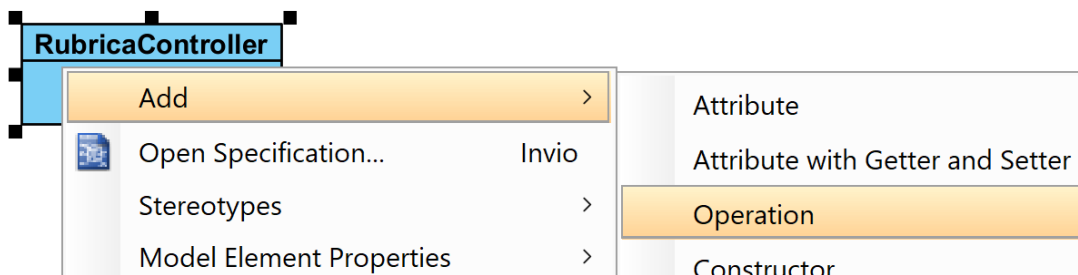
Dalla **Palette** a sinistra, possiamo visualizzare tutti gli elementi UML supportati per la modellazione di un diagramma delle classi. Trasciniamo la voce **Class** nella sezione vuota del diagramma delle classi appena creato, e rinominiamola in **“RubricaController”**, facendo doppio click su di essa. Nel contesto software, il controller è un pattern (cioè una soluzione consolidata) utilizzato per intercettare gli eventi esterni a un sistema (ad esempio, una richiesta utente che ha origine da una interfaccia grafica) e inoltrarli alle classi responsabili della loro gestione, pertanto le classi di tipo controller sono comuni nella modellazione.



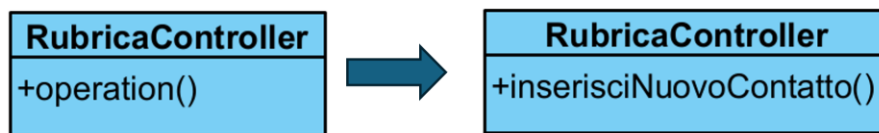
Rileggendo la descrizione dell'operazione **“inserisciNuovoContatto”** a inizio documento, possiamo assumere anche l'esistenza di una classe che rappresenta la *rubrica* e di una classe che rappresenta il *contatto* da inserire. Introduciamo quindi due nuove classi nel diagramma, chiamate “Rubrica” e “Contatto”, come fatto precedentemente.



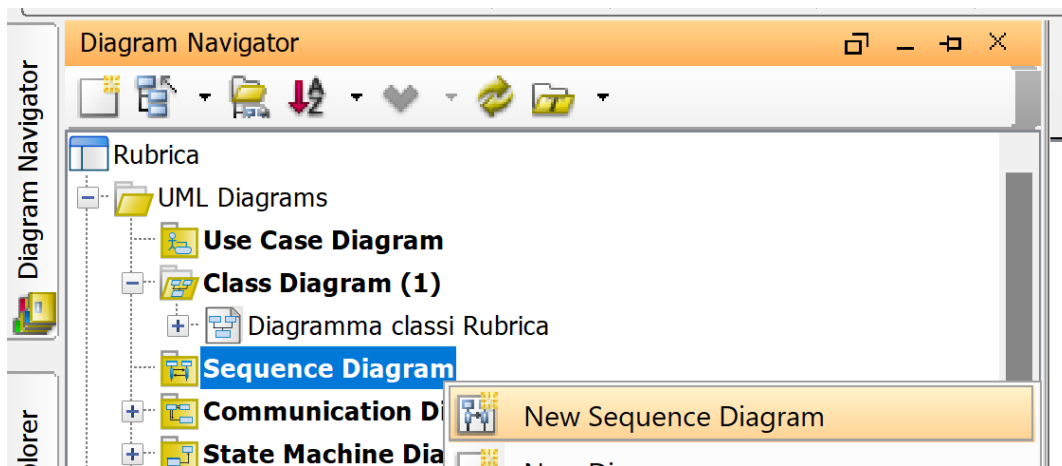
Ora aggiungiamo un'operazione alla classe “RubricaController”, chiamata `inserisciNuovoContatto()`, che ci sarà utile come operazione iniziale per gestire la richiesta dell'utente di inserire un contatto. Per inserire un'operazione nella classe “RubricaController”, clicchiamo col tasto destro su di essa, quindi selezioniamo **Add > Operation**.



Ciò aggiungerà un'operazione senza nome associata alla classe. Facendo doppio click sul nome dell'operazione, rinominiamola "inserisciNuovoContatto".



Al momento non aggiungiamo altri dettagli al diagramma delle classi. Creiamo invece un diagramma di sequenza per modellare l'aspetto dinamico relativo all'inserimento di un nuovo contatto in Rubrica. Da **Diagram Navigator**, clicchiamo col tasto destro su **Sequence Diagram**, quindi selezioniamo **New Sequence Diagram**.

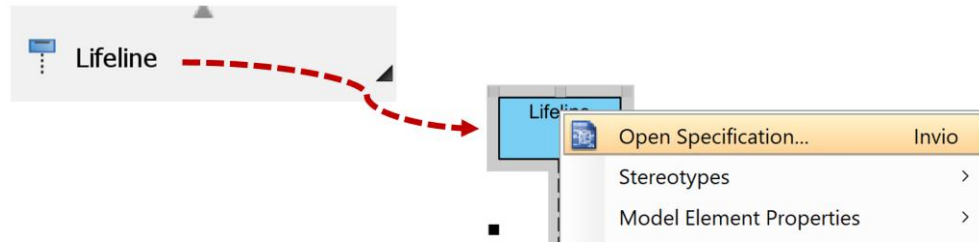


Rinominiamolo come "SD InserisciNuovoContatto", facendo doppio click sull'etichetta rappresentate il nome. Otterremo il seguente risultato.

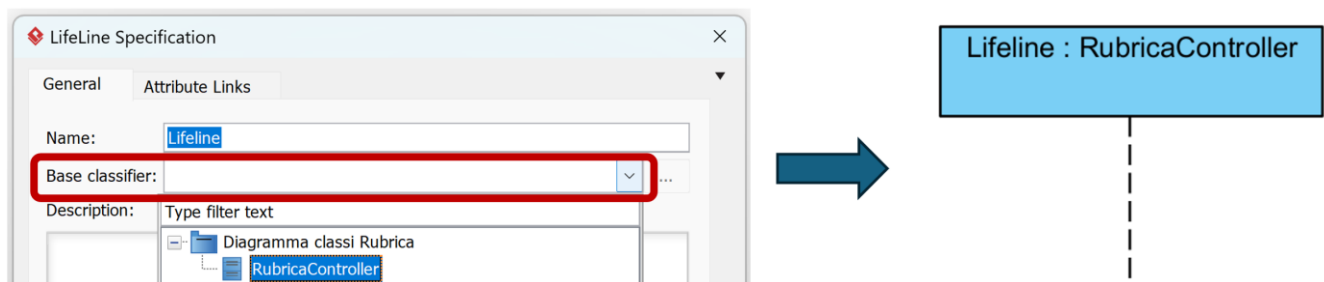


Ora possiamo introdurre una lifeline relativa al nostro punto di accesso al sistema nel diagramma di sequenza appena creato, che sarà il nostro controller. Selezioniamo dalla **Palette** a sinistra, ora aggiornata con l'elenco di elementi UML a supporto dei diagrammi di sequenza, la voce **Lifeline** e trasciniamola nel diagramma; quindi, clicchiamo su di essa col tasto destro del mouse e selezioniamo la voce **Open Specification**.

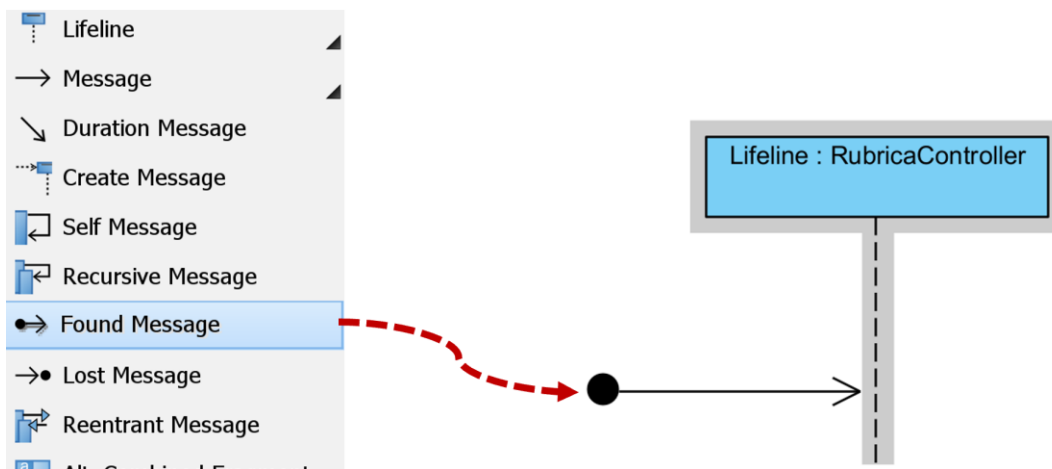




Nella finestra che si aprirà, espandiamo le voci sotto **Base classifier**, selezioniamo “RubricaController” sotto la voce “Diagramma classi Rubrica”, e confermiamo. Ciò collegherà dal punto di vista del modello UML la lifeline alla classe precedentemente creata, permettendo di far riferimento alle sue operazioni durante la modellazione del diagramma di sequenza. Il risultato, adattando la dimensione della lifeline in modo che il testo sia interamente visibile, è il seguente.



Come già detto, la lifeline “RubricaController” farà da mediatrice tra gli eventi esterni e le classi più interne del sistema. Pertanto, per rappresentare un evento esterno diretto a “RubricaController” introduciamo un **messaggio trovato** (anche chiamato **found message**). Clicchiamo sulla voce **Found Message** nella **Palette**, quindi trasciniamo a partire da uno spazio vuoto del diagramma fino a toccare la lifeline, come mostrato di seguito.

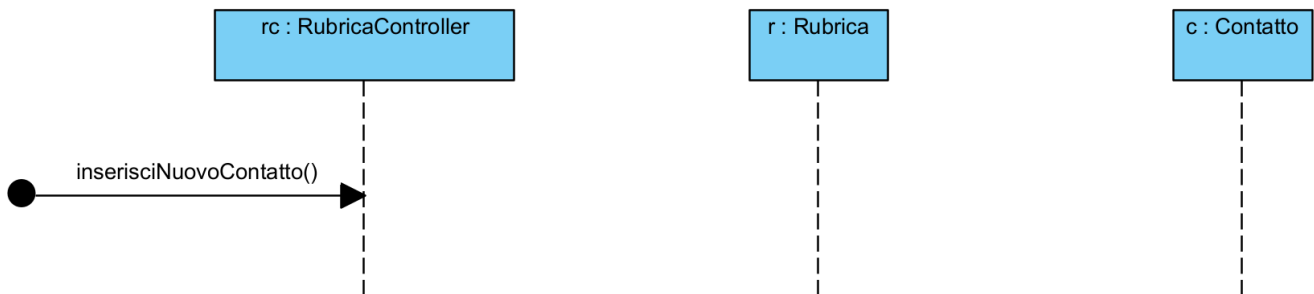


Visual Paradigm proporrà la voce `inserisciNuovoContatto()` da associare al messaggio, che selezioneremo. Ciò è possibile perché abbiamo precedentemente creato l’operazione all’interno della classe “RubricaController” nel diagramma delle classi, nonché collegato la lifeline alla classe. Pertanto, per ogni messaggio entrante nella lifeline, VP proporrà la lista di operazioni associate. Il **messaggio trovato**, che giunge dall’esterno a seguito della richiesta dell’utente,

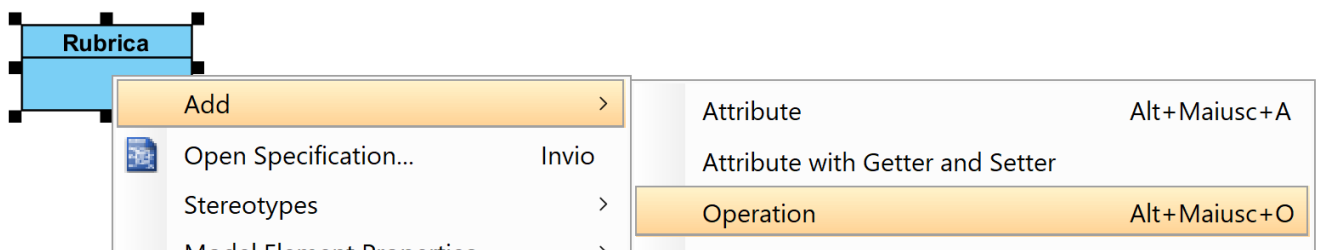
invokerà dunque l'omonima operazione incapsulata in "RubricaController", attivando così il diagramma di sequenza.



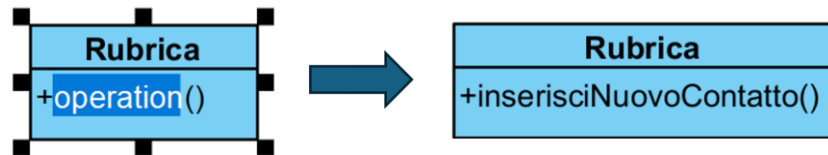
Ora aggiungiamo due lifeline, una per "Rubrica" e una per "Contatto". Una volta aggiunte, come fatto per "RubricaController", via tasto destro > **Open Specification**, selezioniamo un **Base classifier** per ciascuna classe, espandendo le voci relative: in particolare, una lifeline andrà associata alla classe "Rubrica" e una alla classe "Contatto". Rinominiamo le lifeline aggiunte in "rc" per "RubricaController", "r" per "Rubrica" e "c" per "Contatto", ottenendo quanto di seguito.



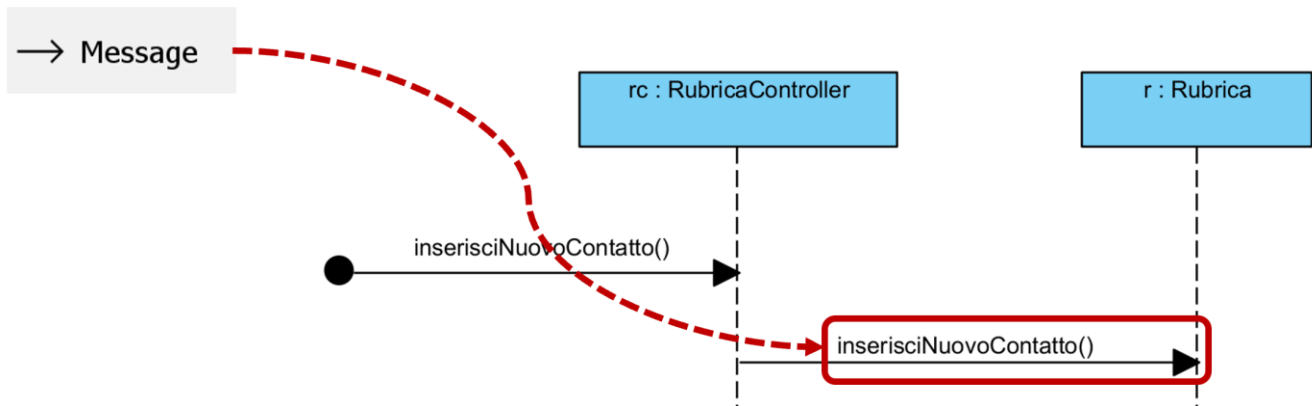
Ora vogliamo aggiungere il messaggio `inserisciNuovoContatto()` da "RubricaController" a "Rubrica", in modo da assegnare la responsabilità di creare contatti a "Rubrica". Come prima cosa, dobbiamo inserire un'operazione nella classe "Rubrica" nel diagramma delle classi. Quindi, via **Diagram Navigator**, torniamo sul diagramma delle classi, selezioniamo la classe "Rubrica" e, cliccando col tasto destro su di essa, selezioniamo **Add > Operation**.



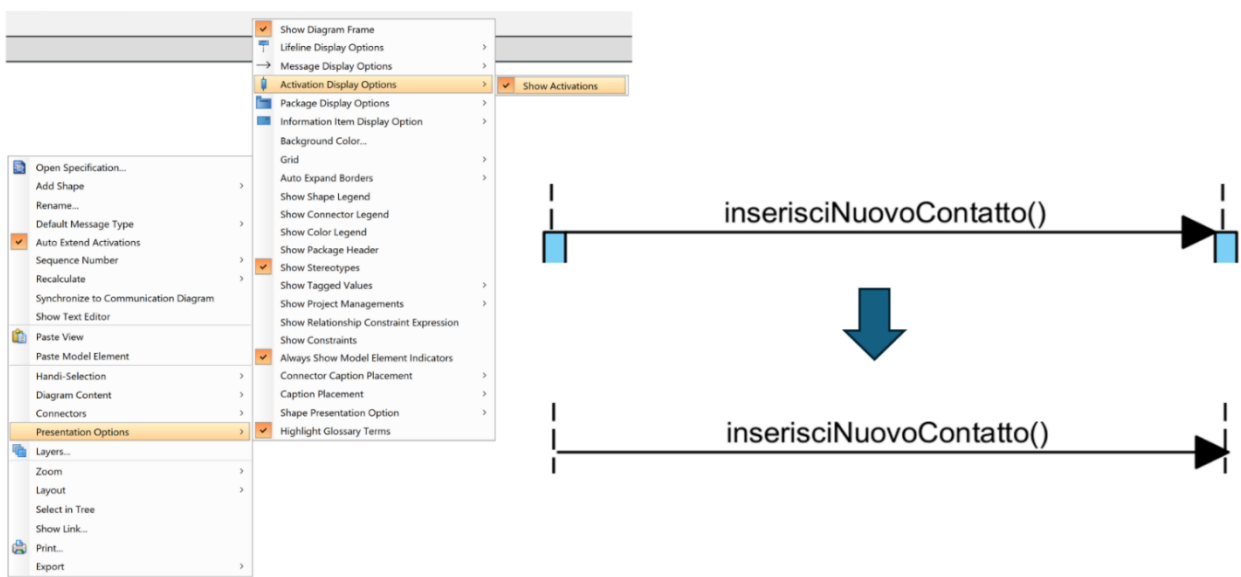
Ciò aggiungerà un'operazione senza nome associata alla classe. Diamo il nome "inserisciNuovoContatto".



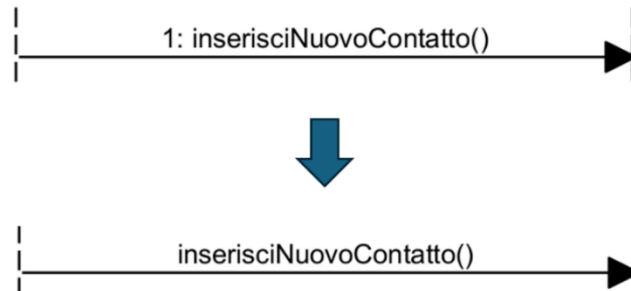
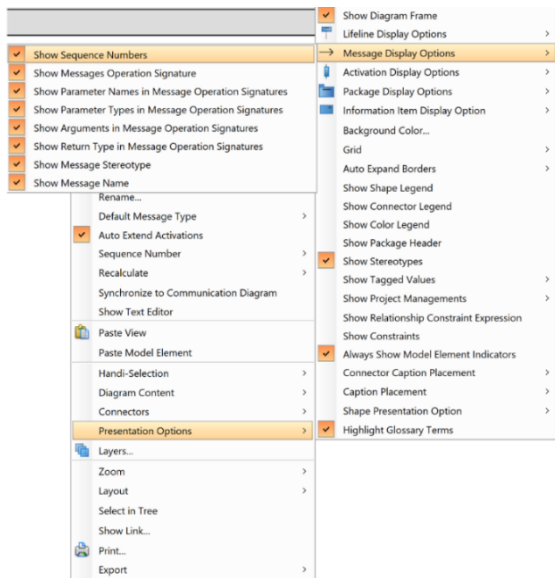
A questo punto possiamo inviare un messaggio da “RubricaController” a “Rubrica”. Torniamo dunque sul diagramma di sequenza e dalla **Palette** selezioniamo la voce **Message**, che rappresenta un messaggio inviato da un’istanza di classe a un’altra. Trasciniamo il mouse da “RubricaController” a “Rubrica”, selezionando l’operazione appena aggiunta. L’effetto sarà quello di produrre un messaggio etichettato con l’operazione desiderata.



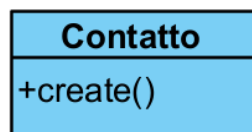
In VP è possibile personalizzare i dettagli visualizzati in un diagramma; ad esempio, si può nascondere il numero sequenziale dei messaggi o le barre di attivazione blu che vengono solitamente create in corrispondenza di un messaggio inserito. Per nascondere questi dettagli (o per visualizzarli nuovamente), si può cliccare con il tasto destro su uno spazio vuoto del diagramma, quindi selezionare **Presentation Options > Activation Display Options > Show Activations** (per visualizzare/nascondere le



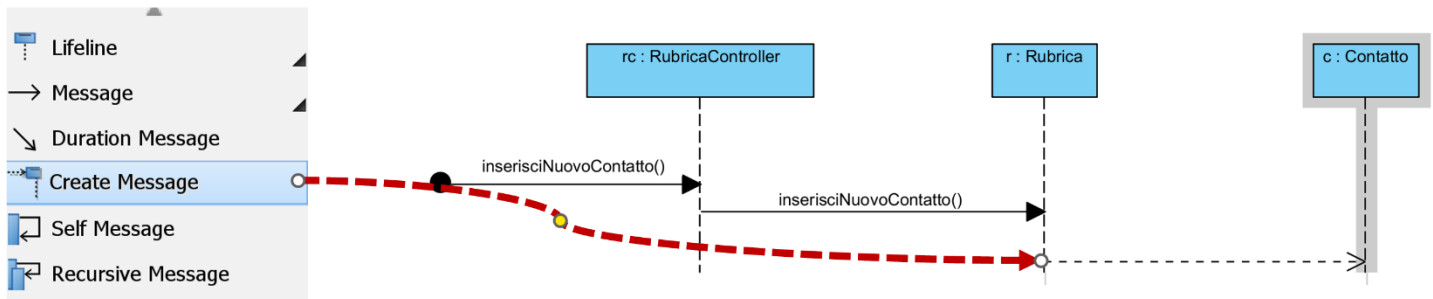
barre di attivazione) oppure **Presentation Options > Message Display Options > Show Sequence Numbers** (per visualizzare/nascondere i numeri associati ai messaggi).



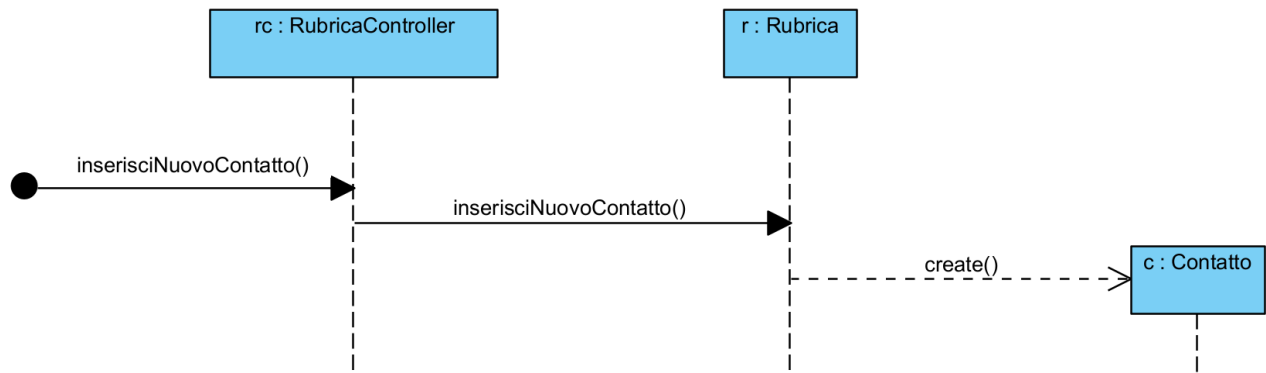
Vogliamo ora aggiungere il **messaggio di creazione** che partirà dalla lifeline “Rubrica” alla lifeline “Contatto”, dato che vogliamo rappresentare la creazione di un nuovo contatto, la cui responsabilità è in mano alla classe “Rubrica”. Il primo passo avviene nuovamente attraverso il diagramma delle classi, dove creeremo un’operazione create() per la classe “Contatto”, come visto in precedenza (tasto destro sulla classe, **Add > Operation**)



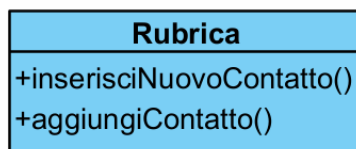
Tornando sul diagramma di sequenza, clicchiamo l'icona **Create Message** dalla **Palette**, quindi trasciniamo da “Rubrica” a “Contatto”, selezionando l’operazione create() che verrà proposta.



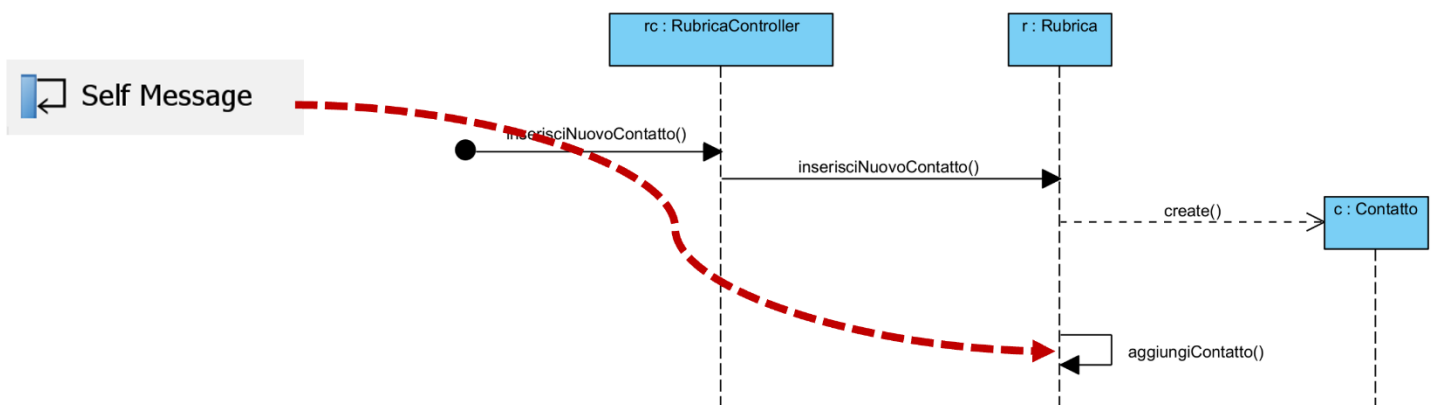
Ciò provocherà un riposizionamento della lifeline “Contatto” all’altezza del messaggio di creazione, a dimostrazione del fatto che l’istanza “c” di “Contatto” esisterà dal momento in cui verrà effettivamente creata.



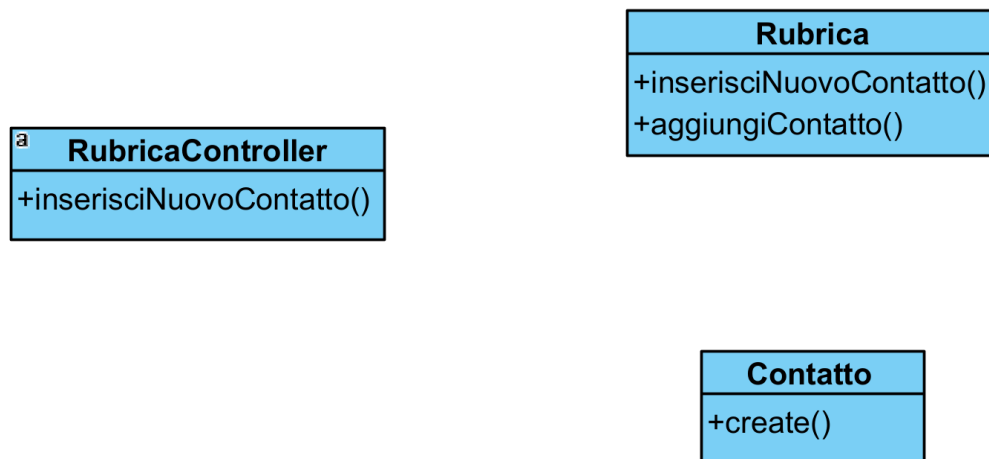
A conclusione del diagramma di sequenza, vogliamo ora aggiungere un messaggio di tipo **self** con origine e destinazione la lifeline “Rubrica”, avente lo scopo di modellare l’aggiunta alla collezione di contatti della rubrica il nuovo contatto appena creato. Prima di fare questo però andiamo sul diagramma delle classi, e inseriamo l’operazione `aggiungiContatto()` alla classe “Rubrica”.



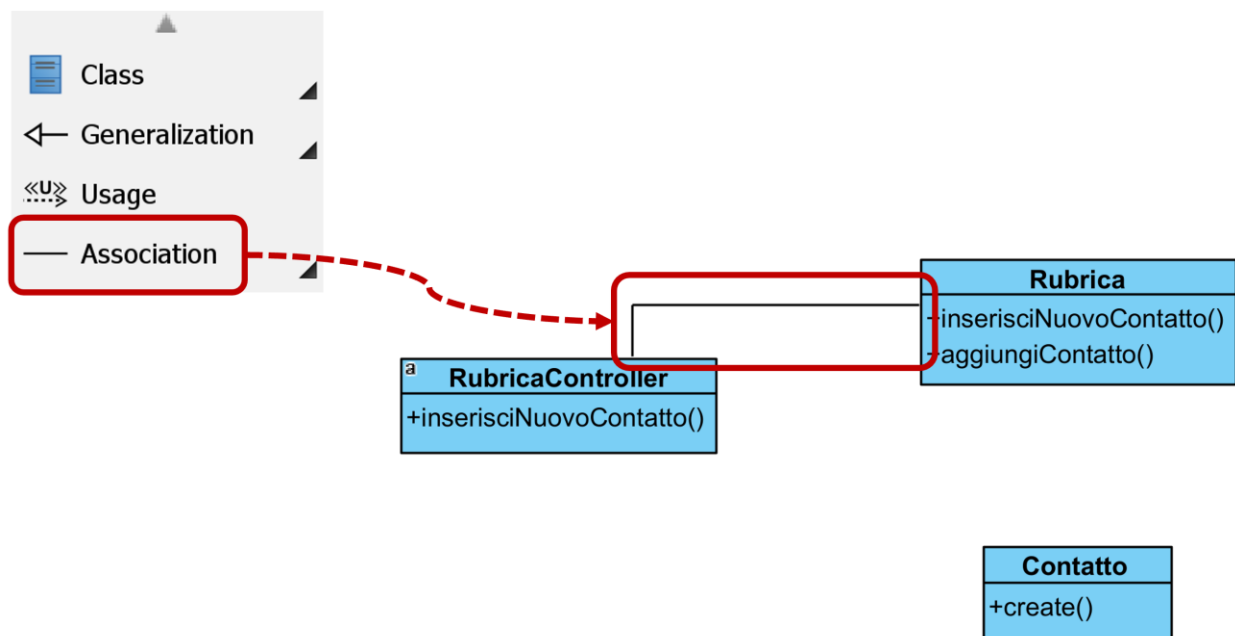
Ora, tornando nel diagramma di sequenza, dalla **Palette** selezioniamo **Self Message** e, senza trascinare ma con un solo click del tasto sinistro del mouse, clicchiamo sopra la linea tratteggiata della lifeline “Rubrica”, subito dopo il messaggio `create()`. Come al solito, verrà richiesto di selezionare un’operazione, quindi indichiamo `aggiungiContatto()`. Il risultato sarà il seguente.



Torniamo ora sul diagramma delle classi, di seguito visualizzato.



Dobbiamo ora completare il diagramma aggiungendo le associazioni. Dalla **Palette** a sinistra selezioniamo la voce **Association**, e trasciniamo da “RubricaController” a “Rubrica”, in modo da creare un’associazione tra le due classi.



Clicchiamo ora col tasto destro del mouse sull’associazione appena creata e selezioniamo **Open Specification**. Quindi, dalla finestra che si apre, selezioniamo `<Unspecified>` per la voce **Navigable** associata alla classe “RubricaController” e `True` per la stessa voce associata a “Rubrica”, e confermiamo.

Association Specification

×

General

▼

Name:

Visibility:

<Unspecified>

▼

Association End From

Role:

...

Element:

RubricaController

...

Multiplicit...

<Unspecified>

▼

Navigable:

<Unspecified>

▼

Association End To

Role:

...

Element:

Rubrica

...

Multiplicit...

<Unspecified>

▼

Navigable:

True

▼

Description:

B

▼

≡

▼

1

2

3

≡

▼

F

▼

▼

▼

*F*

▼

+

▼

▼

▼

▼

☐ Abstract

☐ Leaf

☐ Derived

☐ Final specialization

Reset

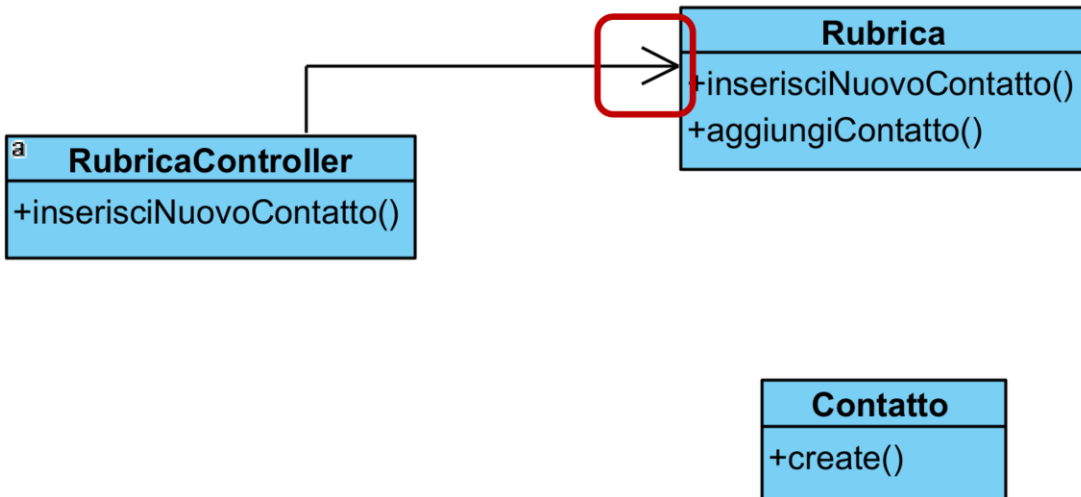
OK

Cancel

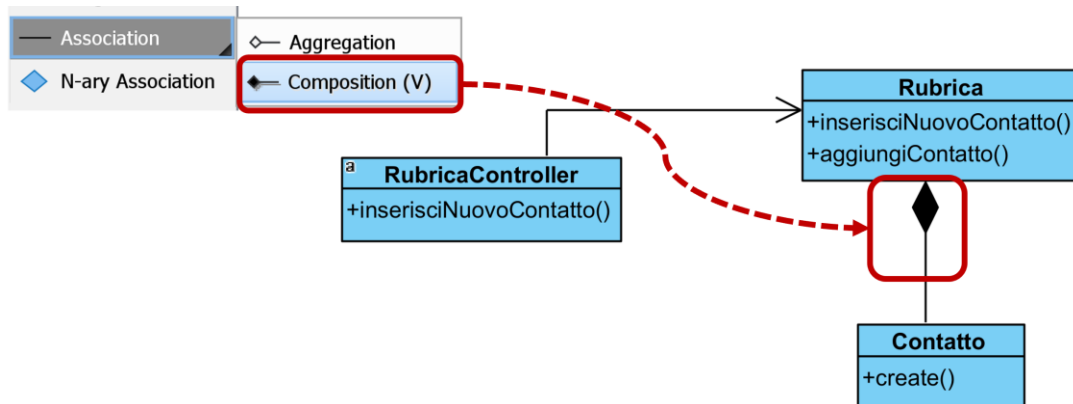
Apply

Help

Questo ci permetterà di esplicitare la navigabilità dell'associazione, che da "RubricaController" va verso "Rubrica" ma non viceversa.



Allo stesso modo, dobbiamo aggiungere l'associazione tra “Rubrica” e “Contatto”. In questo caso non si tratta però di una semplice associazione: una rubrica è composta da molti contatti e un contatto appartiene a un'unica rubrica – e non può esistere in assenza di essa. Abbiamo quindi bisogno di una **composizione** (diamante nero). Dalla **Palette** clicchiamo sul triangolino nero di fianco alla voce **Association**, per espandere le opzioni, e selezioniamo **Composition**, trascinando da “Rubrica” a “Contatto”.

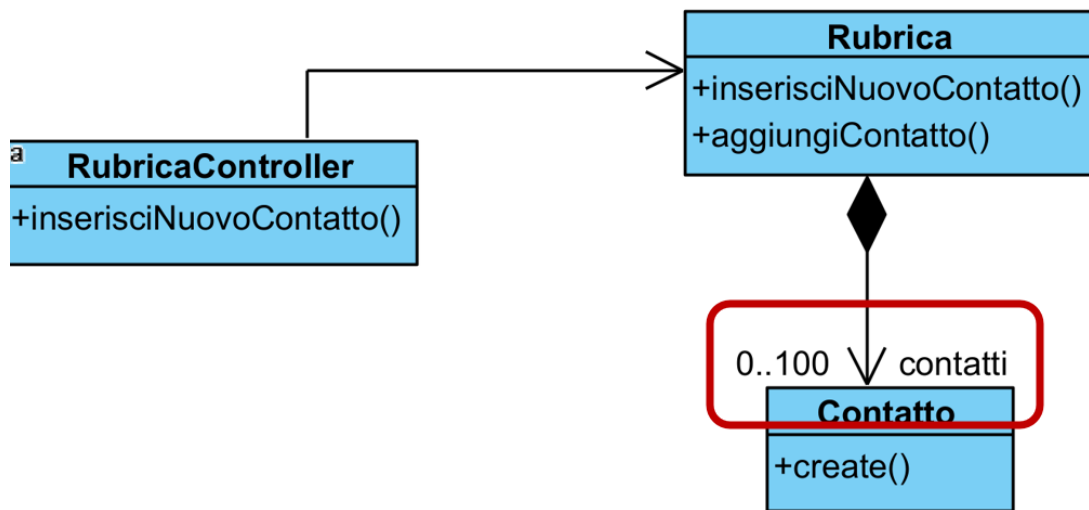


Ricordandoci che una rubrica può avere da 0 a 100 contatti, come da descrizione del sistema, selezioniamo la composizione appena aggiunta nel diagramma e, sempre dalla finestra aperta via tasto destro > **Open Specification**: selezioniamo `<Unspecified>` per la voce **Navigable** associata a “Rubrica”, inseriamo “contatti” nel campo **Role** associato a “Contatto”, e modifichiamo la voce **Multiplicity** dal lato “Contatto” inserendo “0..100”.

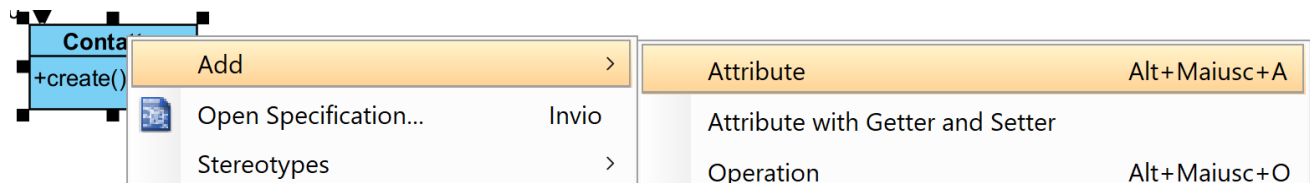
The screenshot shows the **Association Specification** dialog box. The **General** tab is selected. The **Name** field is empty. The **Visibility** is set to `<Unspecified>`. The **Association End From** section shows **Role** as empty, **Element** as **Rubrica**, **Multiplicity** as `<Unspecified>`, and **Navigable** as `<Unspecified>`. The **Association End To** section shows **Role** as **contatti**, **Element** as **Contatto**, **Multiplicity** as **0..100**, and **Navigable** as **True**. The **Description** section has a text area with a rich text editor toolbar. At the bottom, there are checkboxes for **Abstract**, **Leaf**, **Derived**, and **Final specialization**, all of which are unchecked. The **Reset**, **OK**, **Cancel**, **Apply**, and **Help** buttons are at the bottom.



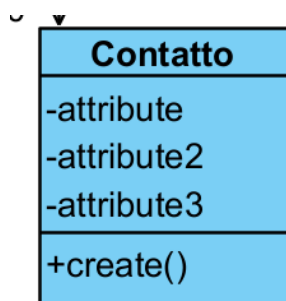
Il risultato sarà il seguente. Si noti che le molteplicità di tipo 1 si possono omettere, in quanto implicite.



Dobbiamo ancora aggiungere gli attributi alla classe “Contatto”. In particolare, aggiungiamo “nome”, “descrizione”, e “protetto”. Un modo per aggiungere attributi a classi è quello di cliccare col tasto destro del mouse sulla classe “Contatto”, quindi selezionare **Add > Attribute**.



Eseguiamo questo passaggio tre volte per creare tre attributi.



Oltre al nome degli attributi, dovremo considerare altre informazioni per essi, quali **visibilità**, **tipo**, **molteplicità** e **valore predefinito**. Clicchiamo sul primo attributo e, aprendo il menu col tasto destro, selezioniamo la voce **Open Specification**. Verrà visualizzata la finestra seguente.

Attribute Specification

General

Name: **nome**

Classifier: Diagramma classi Rubrica.Contatto

Initial value:

Multiplicity: <Unspecified> ☐ Ordered ☒ Unique

Visibility: private

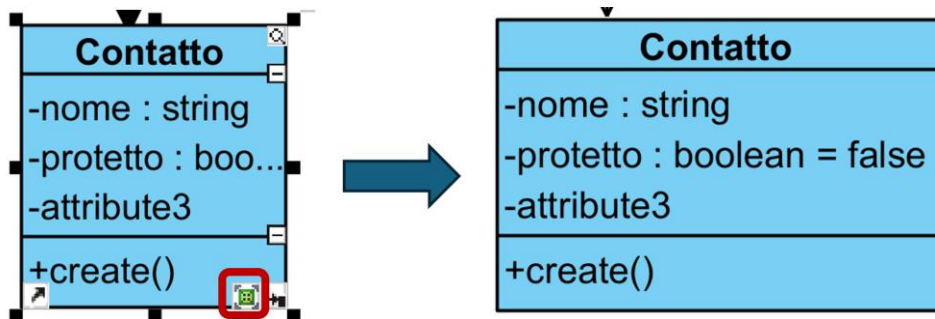
Type: **string**

Type modifier: <Unspecified>

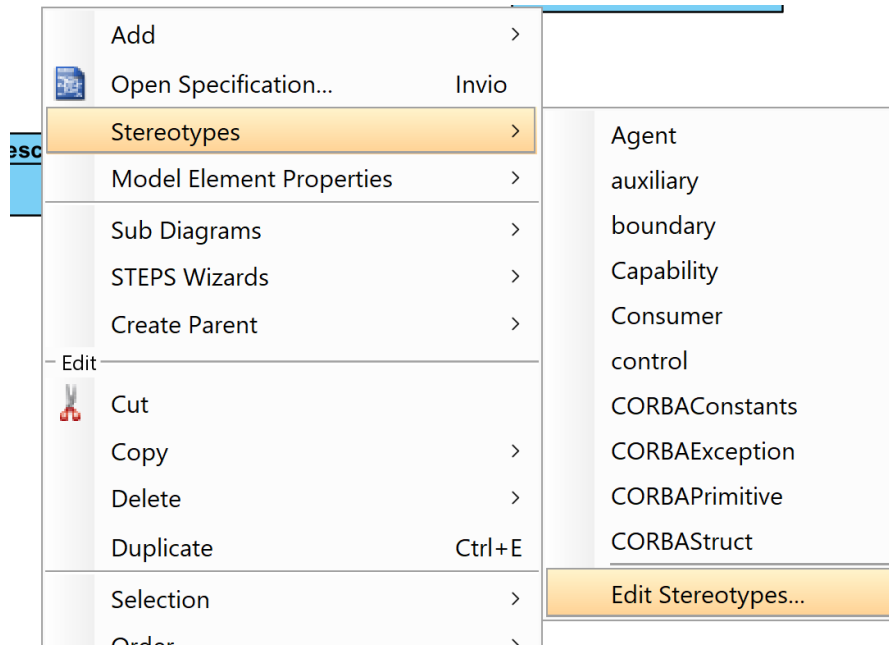
Scope: instance

Aggregation: None

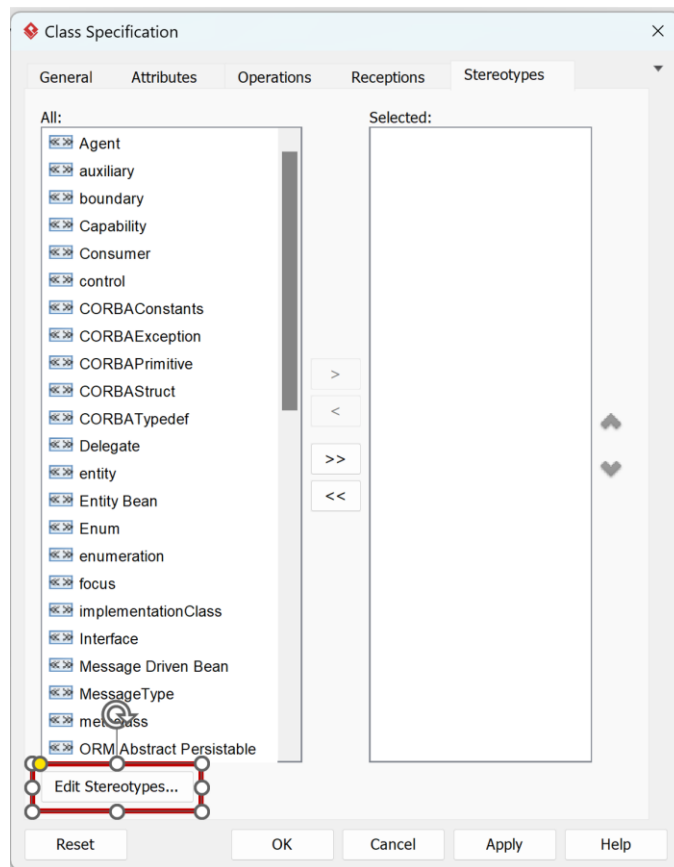
Per quanto riguarda il campo **Name** inseriamo la stringa “nome”, mentre per il campo **Type** selezioniamo il tipo “string” dal menu. Non ci serve invece modificare la molteplicità (**Multiplicity**, di default impostata a 1), la visibilità (**Visibility**, di default impostata su private per gli attributi) o il valore predefinito dell’attributo (**Initial value**, di default vuoto). Ripetiamo il procedimento per il secondo attributo “protetto”: come **Name** inseriamo “protetto”, come **Type** indichiamo boolean, e come **Initial value** inseriamo il valore “false”. Confermiamo e, una volta chiusa la finestra, clicchiamo sul bottone verde in basso a destra nella classe per espandere la visualizzazione altrimenti solo parziale.



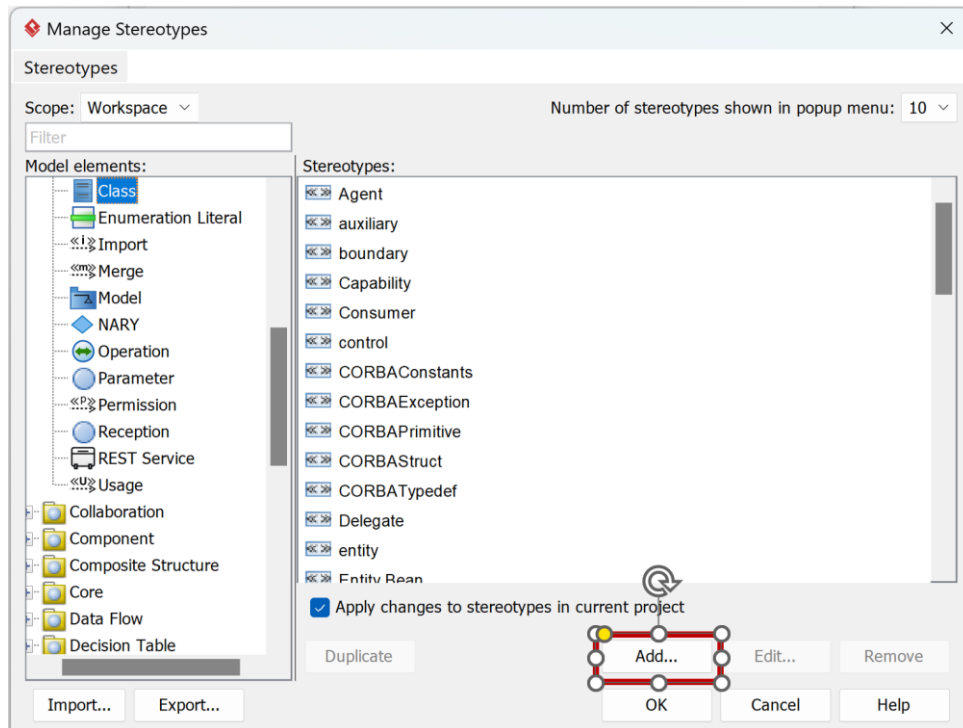
Rimane da gestire l’attributo “descrizione”. Ai fini didattici, assumiamo che la descrizione di un contatto non sia una semplice stringa, ma qualcosa di più complesso: introduciamo un **tipo di dato** (anche chiamato **datatype**) mediante *stereotipi* (un concetto UML usato per personalizzare un modello). Dalla **Palette** a sinistra trasciniamo una nuova classe nel diagramma e diamole il nome “Descrizione”. Quindi, cliccando col tasto destro del mouse sulla classe, selezioniamo **Stereotypes > Edit Stereotypes**.



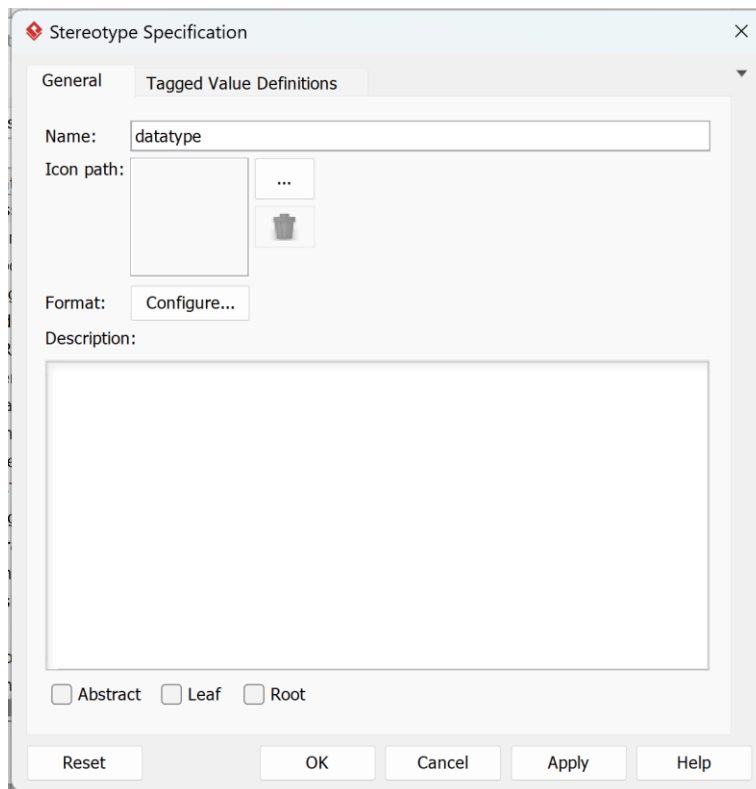
Dalla finestra che si apre, clicchiamo sul bottone **Edit Stereotypes**.



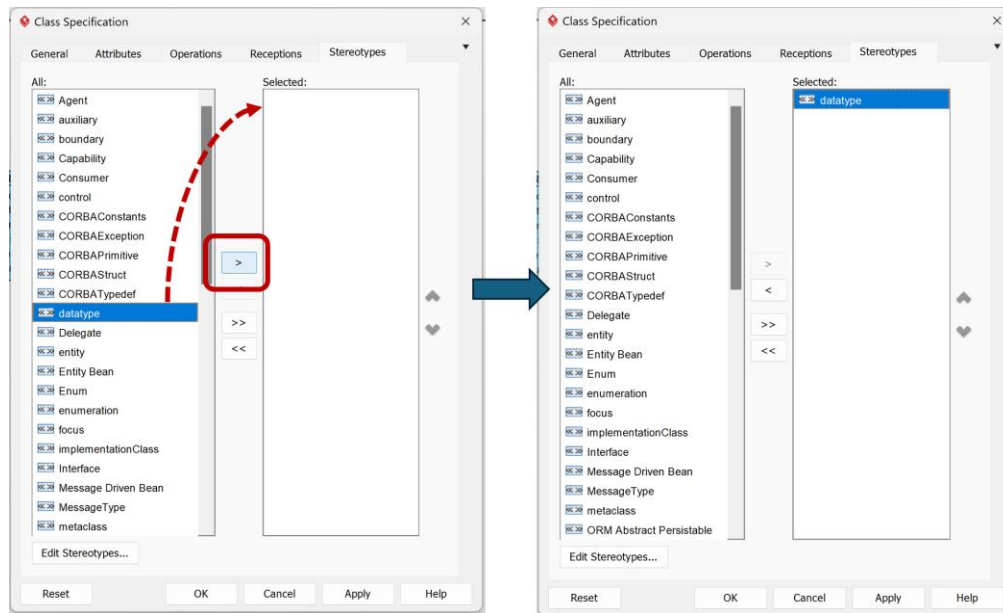
Nella nuova finestra, clicchiamo **Add**.



Nella nuova finestra, digitiamo “datatype” nel campo **Name** e confermiamo.



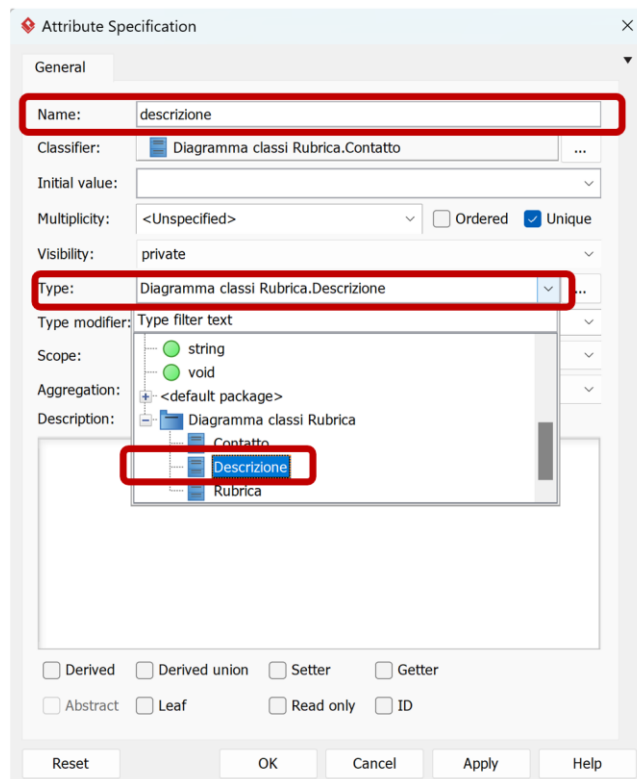
Tornando nella finestra che elenca gli stereotipi, selezioniamo la voce **datatype** appena aggiunta nella lista a sinistra, clicchiamo su > per spostarla nella lista a destra, quindi confermiamo.



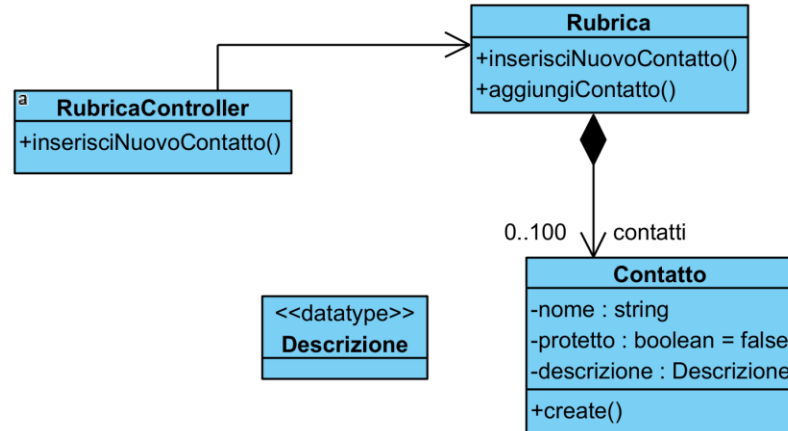
Ciò aggiungerà uno stereotipo <<datatype>> associato alla classe “Descrizione”.

<<datatype>>  
Descrizione

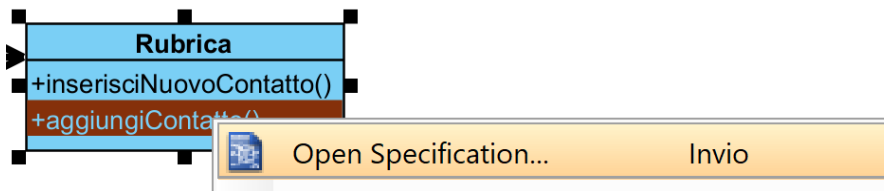
A questo punto, cliccando col tasto destro sull'ultimo attributo della classe “Contatto”, selezioniamo **Open Specification**, e dalla finestra che si apre inseriamo: “descrizione” nel campo **Name** e, espandendo le voci sotto **Type**, selezioniamo il datatype “Descrizione” introdotto.



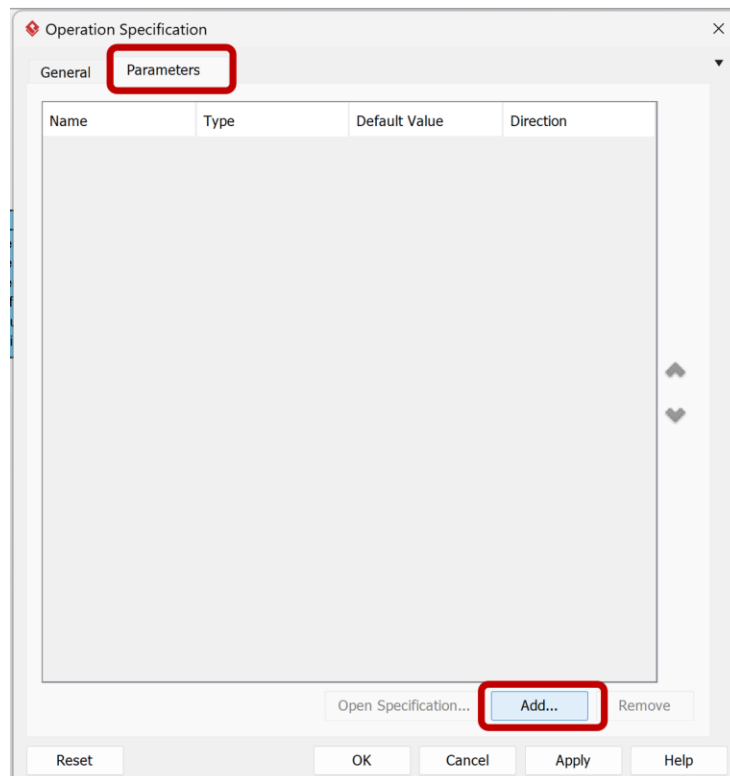
A questo punto il diagramma delle classi è il seguente.



Rimane da definire un parametro per l'operazione `aggiungiContatto()` di "Rubrica", per specificare il contatto da aggiungere. Clicchiamo col tasto destro su `aggiungiContatto()`, quindi **Open Specification**.

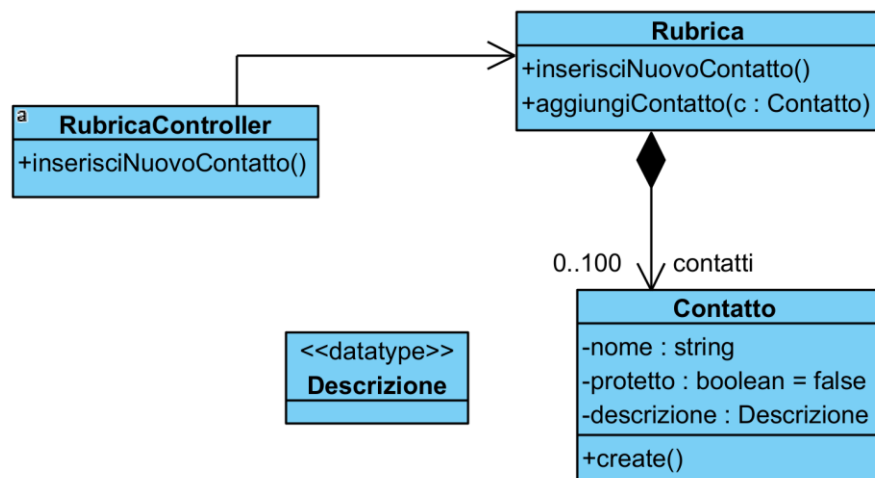


Dalla finestra che si apre, selezioniamo la scheda **Parameters**, quindi clicchiamo su **Add**.

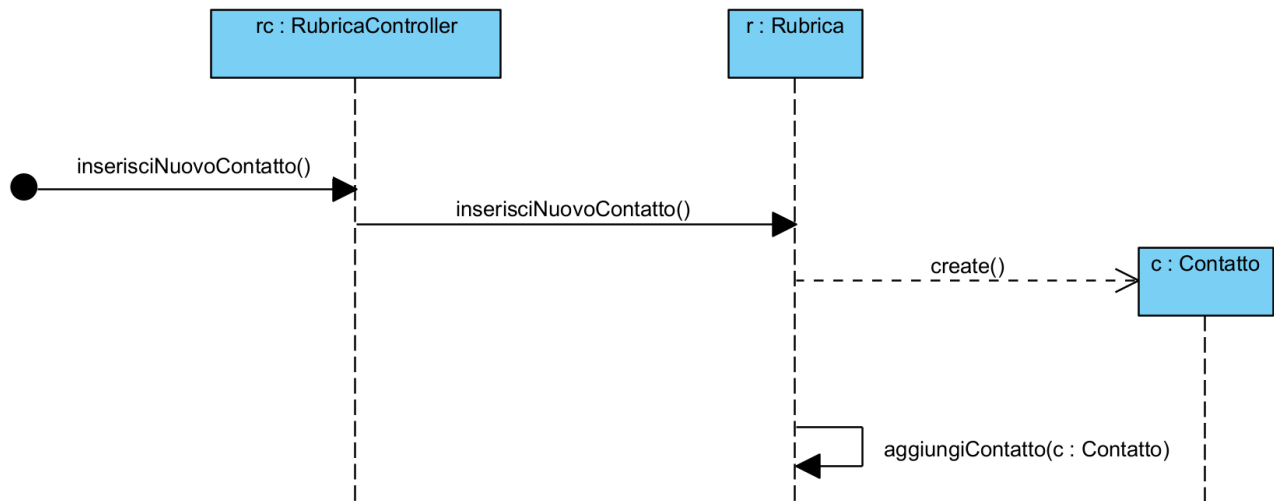


Questo passaggio inserirà un parametro all'operazione aggiungiContatto(). Ragionando su quanto già modellato, noi vogliamo un parametro "c" di tipo "Contatto", in quanto si tratta del contatto creato che vogliamo aggiungere all'insieme di contatti appartenenti alla rubrica. Al parametro appena creato assegniamo il nome "c" modificando il campo **Name**, clicchiamo nel campo **Type** fino a individuare la voce "Contatto", e confermiamo, come di seguito.

Il risultato finale è il seguente.



Per concludere, torniamo sul diagramma di sequenza e osserviamo come il messaggio **self** aggiungiContatto() nella lifeline Rubrica sia stato automaticamente aggiornato con il parametro appena aggiunto all'operazione omonima, dal momento che si lavora in un contesto di modello UML, dove gli elementi coinvolti nella modellazione sono condivisi tra tutti i diagrammi.



Abbiamo finalmente completato la modellazione dell'operazione di aggiunta di un contatto vuoto in rubrica. Si noti che esistono molti modi alternativi, più o meno complessi, per modellare la stessa operazione.