Design model

Introduzione

Obiettivi:

- Definire object-oriented analysis e design
- Illustrare un esempio relativo

Conoscere un linguaggio di programmazione ad oggetti, come Java, è uno step necessario ma non sufficiente per la creazione di un sistema. Apprendere come *pensare ad oggetti* sicuramente non rappresenta un passaggio semplice, ma favorisce alla creazione di software robusti, ben progettati e manuntenibili, caratteristiche esterne ed interne che valorizzano al massimo lo sviluppo e l'implementazione di un progetto.

Tutti i passi che richiedono la metamorfosi di un *domain model* ad un *object-oriented model* non sono mai componenti di un procedimento meccanico, poichè, giunti a tale punto, è stata già effettuata un'analisi del problema e dei requisiti essenziali che dovranno essere poi riproposti all'interno della soluzione. Per cui si riconosce ciò che il sistema dovrebbe fare, ora occorre progettare un procedimento automatico che compia ogni attività voluta.

Il termine *design* sottolinea una *soluzione concettuale* che rientri appieno nei requisiti, ancora prima di una vera e propria implementazione; per cui conclusi diagrammi comportamentali, come *use case* oppure *activity diagram*, e diagrammi strutturali, come *class diagram*, la fase successiva consiste nella definizione degli oggetti software e al modo in cui essi interagiscono per poter soddisfare tutte le richieste.

Modello decisionale

Spesso è adottato un modello decisionale ispirato alle specifiche del framework *Unified Process*. L'elenco funzionale seguente rappresenta un esempio a cui ispirarsi, una semplificazione, dato che per progetti sempre più complessi e dettagliati il modello potrebbe racchiudere aspetti sempre più specifici e sottili.

Un tipico approccio risulta:

- Definizione dei *casi d'uso*. La progettazione di sistemi software nasce dall'esigenza di certi requisiti funzionali, i quali possono essere descritti sintatticamente mediante il *diagramma dei casi d'uso*, rispettando tutte le condizioni del contesto adottato.
- Definizione dei *diagrammi di interazione*. Il passaggio da una qualsiasi modellazione al *Object-Oriented design*, avviene per formalizzare oggetti software e le loro interazioni. Una soluzione spesso adottata consiste nell'uso del *sequence diagram*, capace di illustrare come gli oggetti interagiscono per mezzo di segnali, articolati in molteplici *lifelines*. Come da regola del *framework*, un diagramma di interazioni deve essere adeguato per ogni *use case* del punto precedente.
- Definizione del *diagramma delle classi*. Stabilito il modello che rappresenti una visualizzazione dinamica delle differenti interazioni, potrebbe essere utile ai fini implementativi

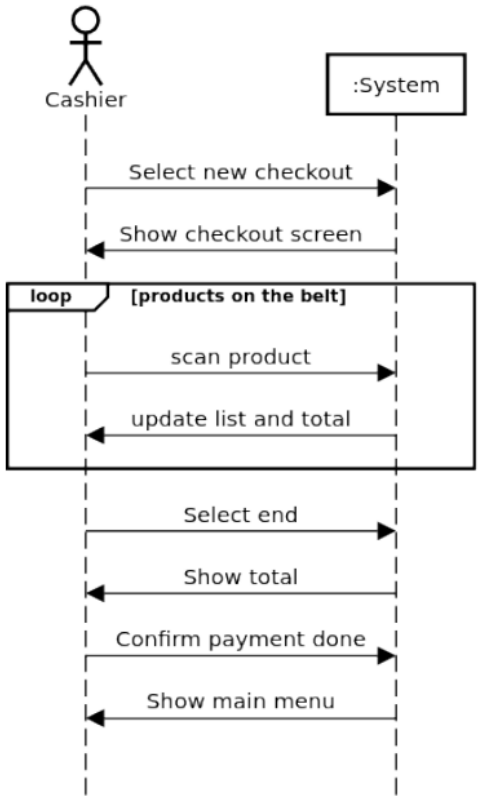
analizzare un approccio statico mediante singole *classi* che caratterizzino gli oggetti. Perciò è adoperato il *diagramma delle classi*, capace di descrivere azioni e attributi inerenti al sistema.

- Dividere il sistema in *interface layer* e in *domain layer*. Questo step rescinde chiaramente il lato amministratore dal lato client, attribuendo differenti finalità rispetto all'utente che si interfacci con il sistema. Le ragioni che spingono tale suddivisione sono relative ad azioni di manutenzione e di modifica al sistema software, in cui lo sviluppatore abbia piena libertà di poter apportare cambiamenti, senza che incidi sull'usufruibilità dell'utente finale.

Esempio
Il *modello decisionale* illustra i passaggi principali che possano indirizzare un'analisi pregressa ad un'implementazione del sistema. Di seguito è riportato il procedimento attuato per illustrare graficamente la transizione da *use case* a *sequence diagram*.

Sono formulati cinque step totali, affinchè siano sufficienti per la costruzione del *diagramma delle classi*, illustrando interazioni e caratteristiche dei differenti soggetti.

Primo step

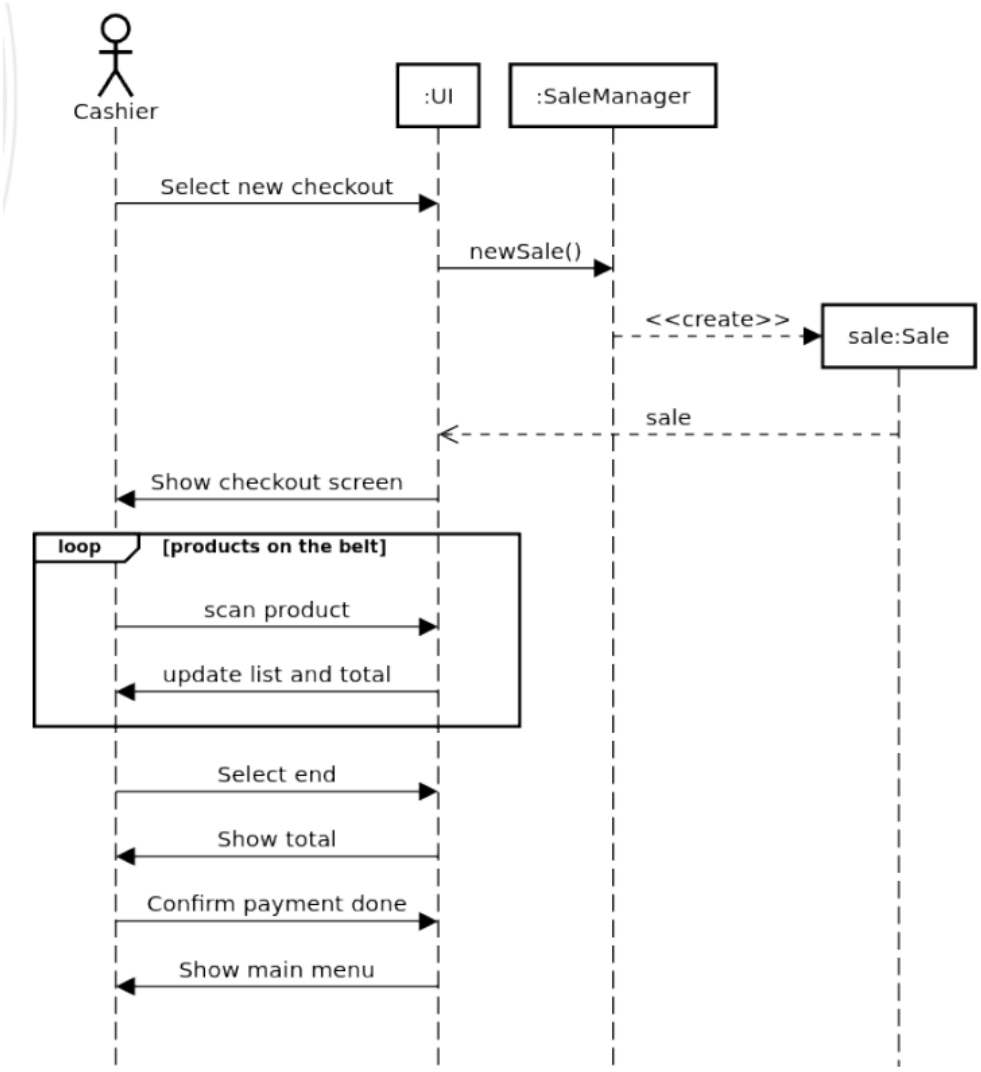


Il modello raffigura l'attore, in questo caso un *cassiere*, che scannerizza i prodotti mediante l'uso del sistema, per poter individuare il prezzo totale che il *cliente* dovrà poi versare.

Individuato l'obiettivo, si percepisce che il diagramma implementato risulti essere un *sequence diagram*, incentrato nella formalizzazione dell'interazioni che gli attori del *behavioral*

diagram si scambiano per mezzo di segnali. Per cui, il primo step illustra una porzione del diagramma comportamentale, stabilendo la totalità dei messaggi necessari affinchè sia possibile rispondere alle richieste dei requisiti funzionali. Una nota particolare è data dalla presenza di un *loop gate*, ossia al suo interno è ripetuta la medesima operazione fino a quando i prodotti appartenenti alla "spesa" non saranno terminati.
Tuttavia un *interation diagram* simile non rispecchia tutte le caratteristiche che possano essere riportate in un seguente *domain model*, perciò occorre inserire ulteriori pattern che analizzino al meglio la struttura delle classi in gioco.

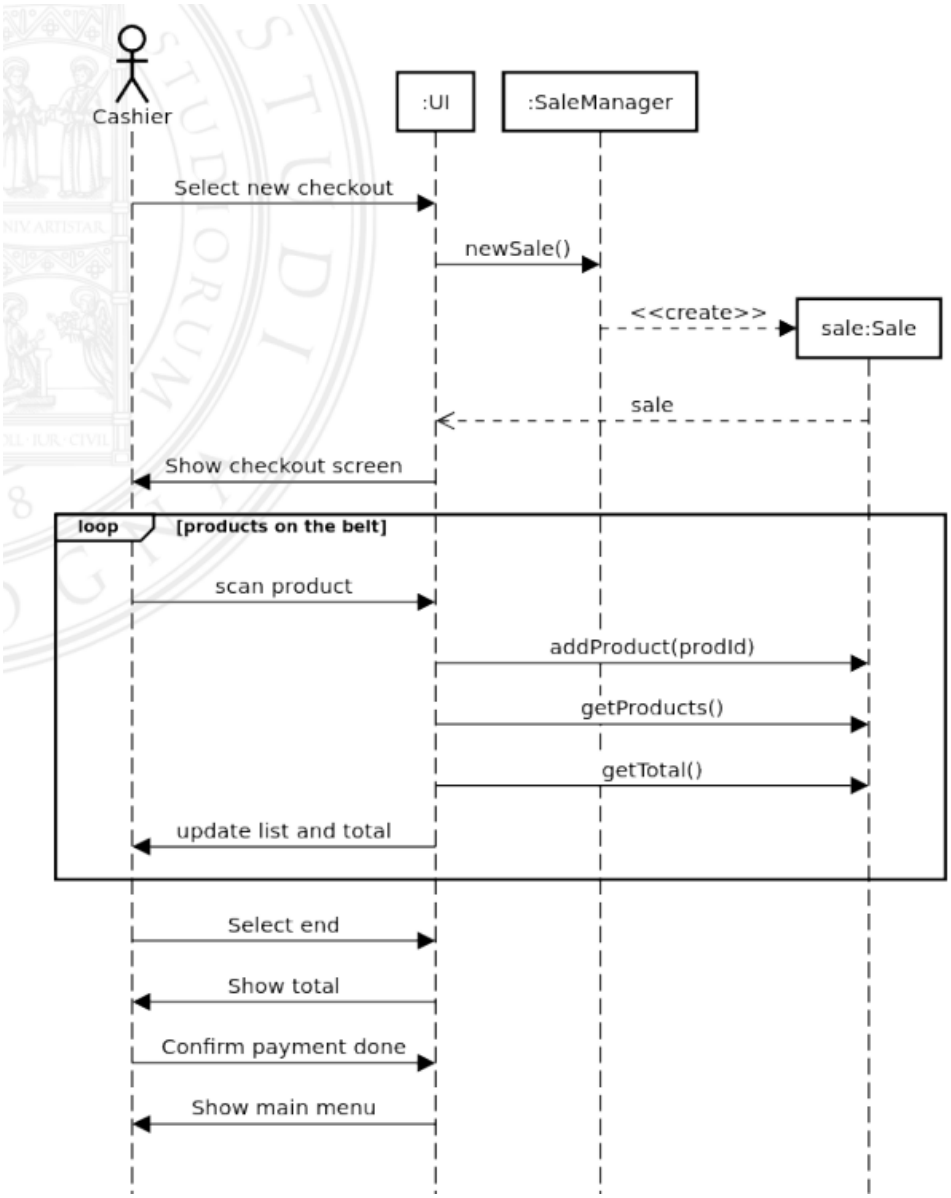
Secondo step



Come da immagine, sono inserite ulteriori istanze. Dato che lo scopo nell'uso di un *sequence diagram* consista nell'individuazione di tutte le classi necessarie per un'implementazione, occorre sfruttare al meglio le capacità appartenenti all'*object-oriented design*, capace di distinguere tutte le dipendenze necessarie. Il *system* originario è stato distinto in due entità, ossia *UI* e *SaleManager*; quest'ultima azione rispecchia la volontà di suddividere il sistema in un *layer interface*, interpellato dal cassiere, e in un *domain layer*, cioè lato amministratore,

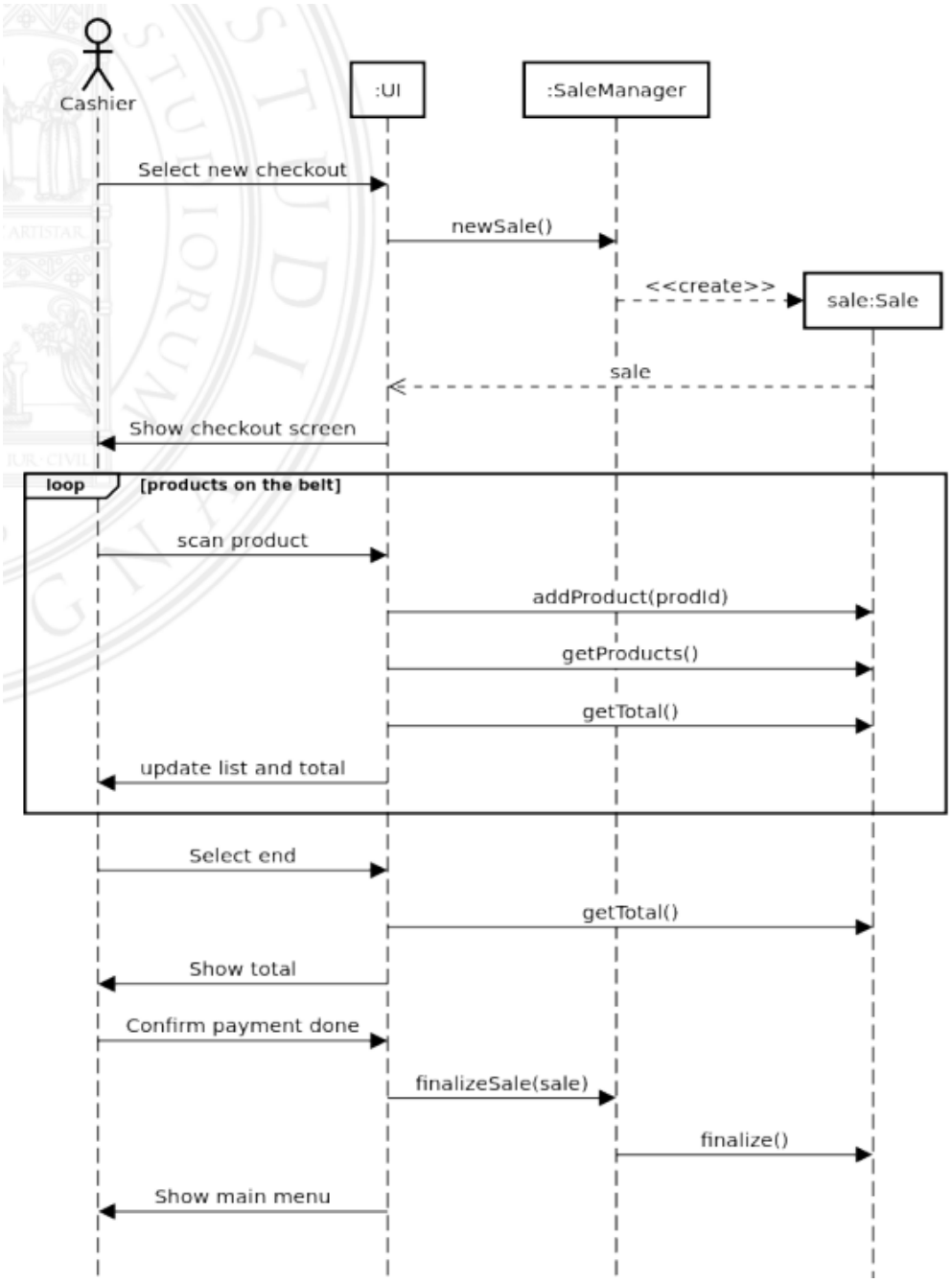
colui che provvederà ad inizializzare una nuova vendita, indicata con il nominativo *sale*, ogni volta sia avanzata una richiesta.

Terzo step



Rispetto alle nuove istanze introdotte, sono inizializzati ulteriori eventi, inerenti alla scannerizzazione di prodotti che compongono la lista. Ad ogni scannerizzazione all'interno dell'oggetto *sale* saranno immessi i prodotti in questione, aggiornando il catalogo della "spesa" effettuata e visualizzando ripetitivamente il costo che dovrà essere sostenuto.

Quarto step



Concluso il *gate loop*, sarà richiesto dall'attore il totale della spesa. Solamente alla conferma del pagamento il cassiere provvederà a interagire con il *SaleManager* per poter finalizzare la vendita da poco avvenuta.

Quinto step
Il *sequence diagram* riportato, è utilizzato per poter modellare solamente due classi del *domain model*. Tuttavia, tramite questo approccio è possibile osservare se l'analisi riportata fino ad ora sia sufficiente o meno per una successiva implementazione. E' importante rispettare

ogni step proposto, per essere certi della correttezza dei diagrammi modellati e affinchè possano essere aggiunti ulteriori nuovi elementi che rispecchino a pieno ogni requisito funzionale.