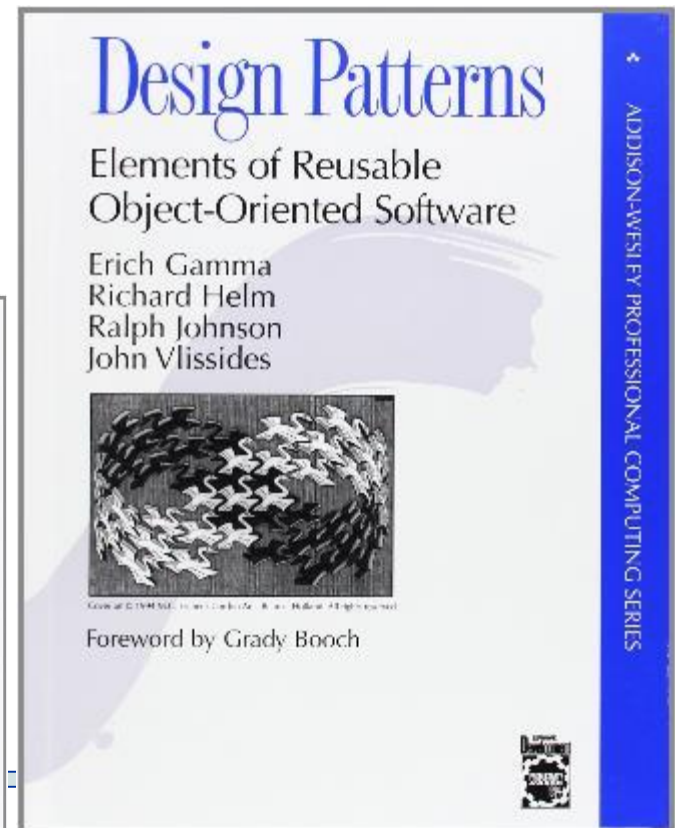
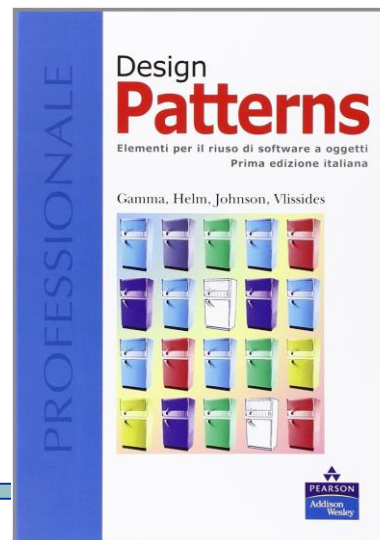


programming pattern

MVC - observer

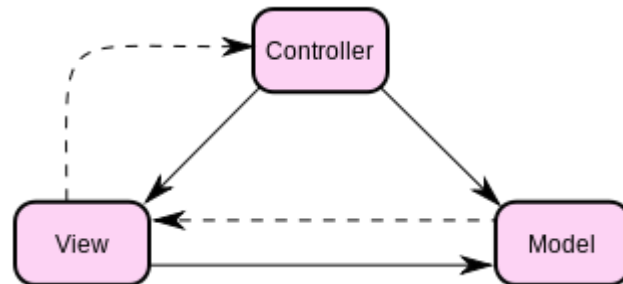
- nell'ambito dell'ingegneria del software, un ***design pattern*** è un concetto che può essere definito "*una soluzione progettuale generale a un problema ricorrente*"
- si tratta di un modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software

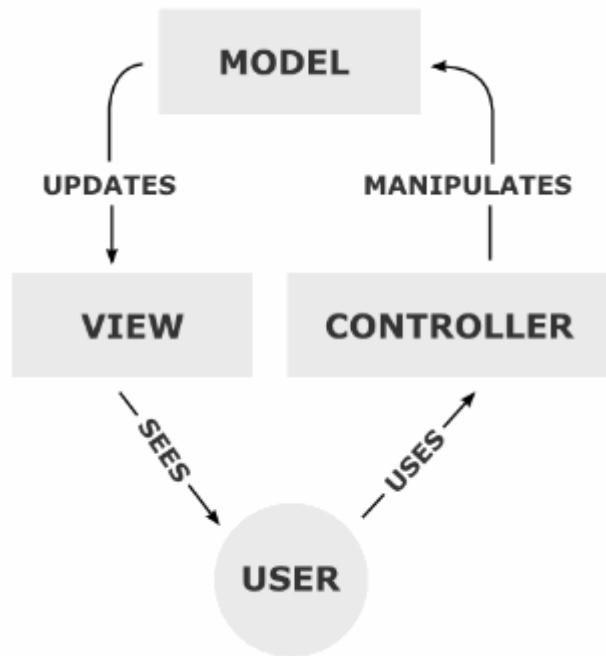
- *Design Patterns:*
Elements of Reusable Object-Oriented Software
 - saggio del 1995
 - Gang of Four (GoF)
 - Erich Gamma
 - Richard Helm
 - Ralph Johnson
 - John Vlissides



- **Model-View-Controller** è un ***pattern*** architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business
- originariamente impiegato dal linguaggio Smalltalk, il pattern è stato sposato da numerose tecnologie moderne, come framework basati su PHP, su Ruby, su Python, su Java (Swing ...), su Objective C o su .NET

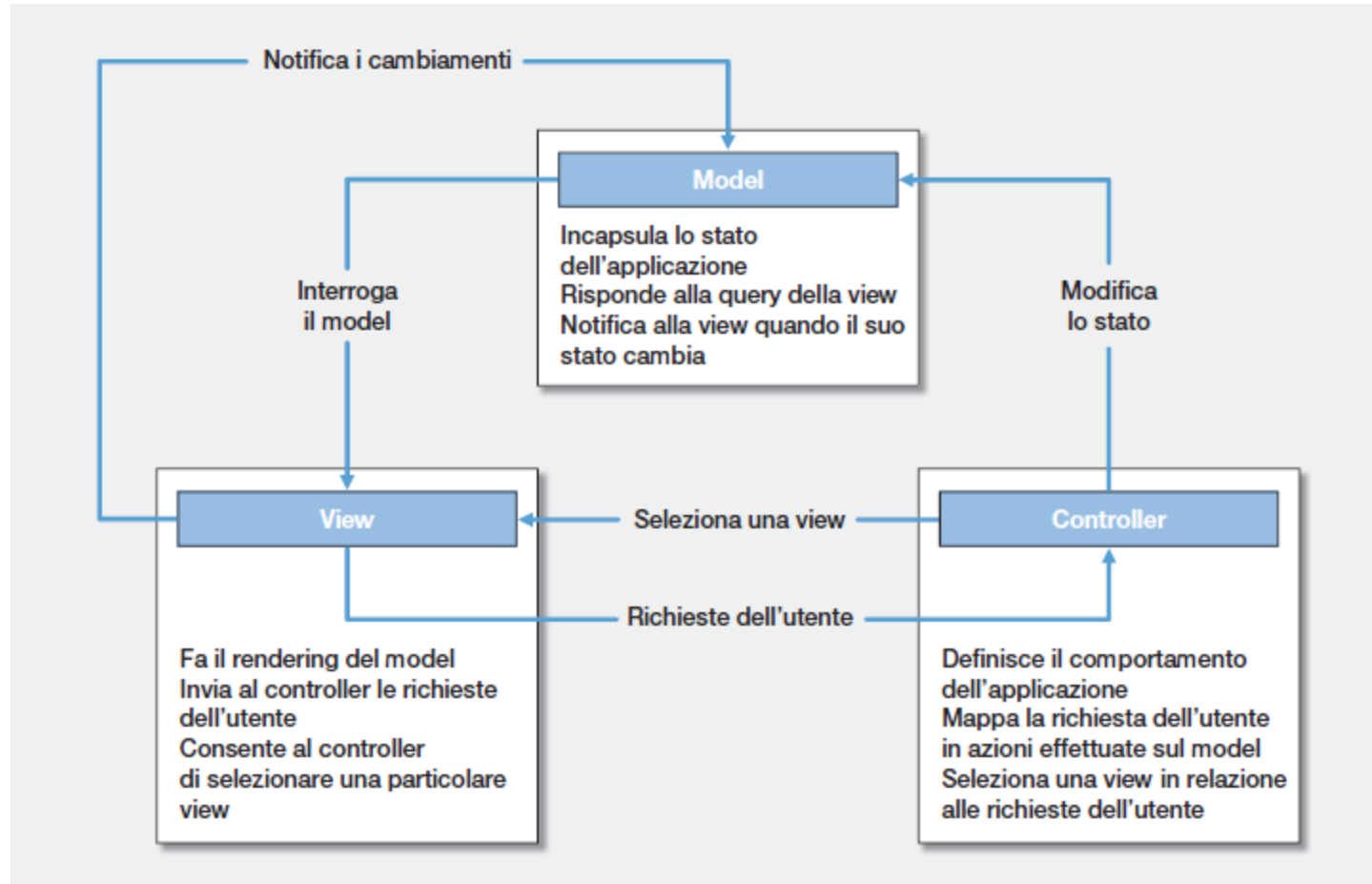
- il pattern è basato sulla **separazione dei compiti** fra i componenti software che interpretano tre ruoli principali:
 - il **model** fornisce i metodi per **accedere** ai **dati** utili all'applicazione
 - il **view** **visualizza** i dati contenuti nel model e si occupa dell'**interazione** con utenti e agenti
 - il **controller** riceve i **comandi** dell'utente (in genere attraverso il view) e li **attua** modificando lo stato degli altri due componenti

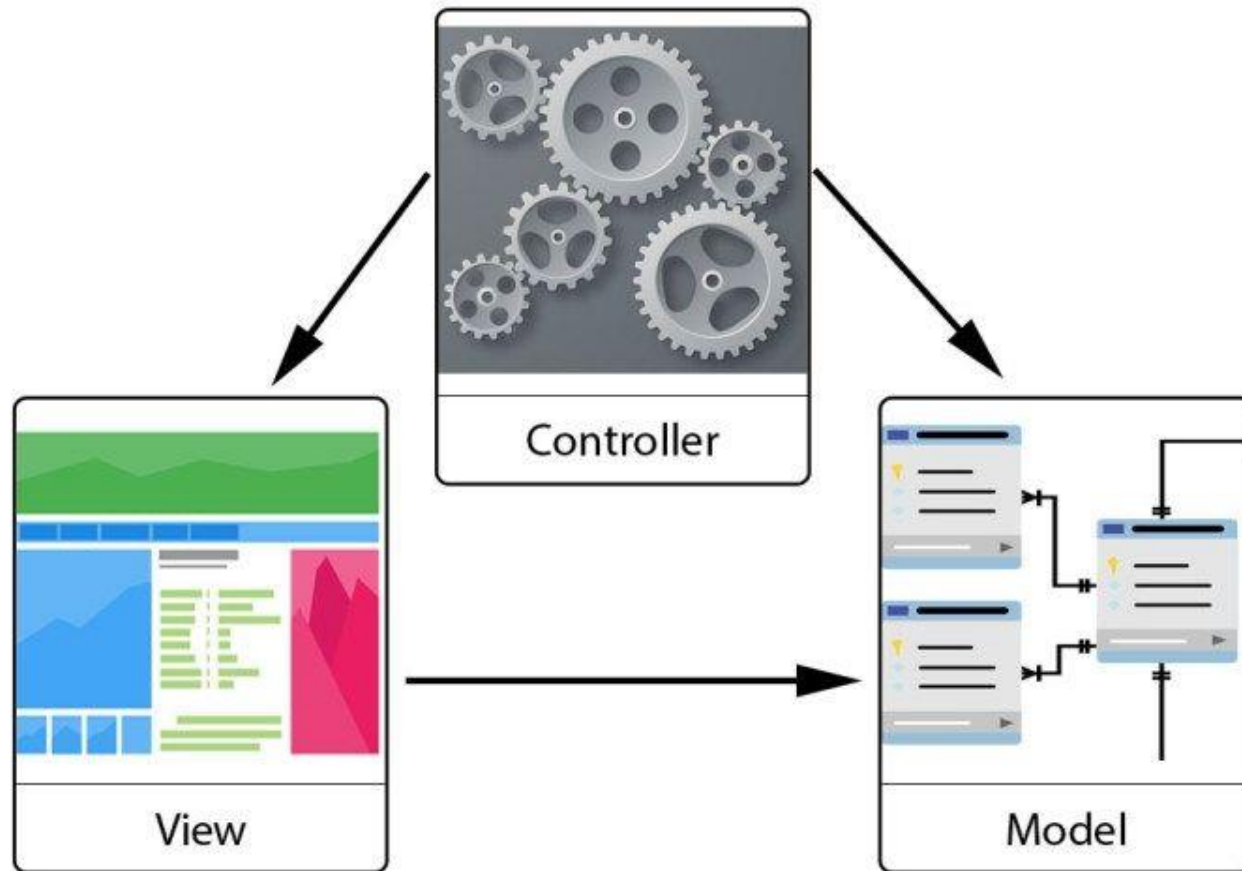




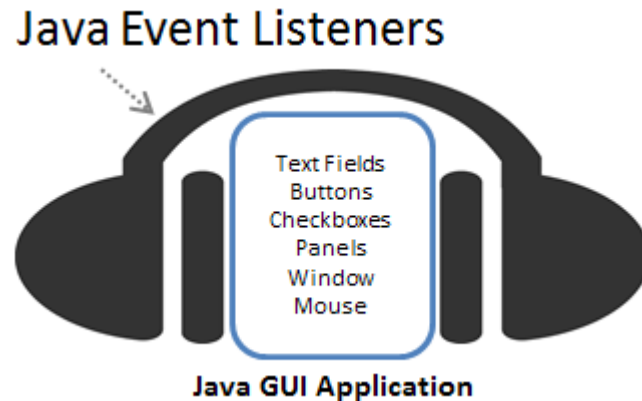
- MVC implica la ***separazione*** fra
 - la ***logica applicativa*** a carico del controller e del model
 - l'***interfaccia utente*** a carico del view

- MVC consente di **suddividere** la *complessità* di realizzazione di un'applicazione dotata di GUI allo scopo di rendere più semplici
 - lo sviluppo
 - la manutenzione
 - la riusabilità del codice
- **indipendenza** tra i business data (model) la logica di presentazione (view) e quella di controllo (controller)
- **viste diverse** per il medesimo model





- programmazione *event driven*:
 - l'esecuzione del codice è guidata dagli eventi generati dall'interazione dell'utente con i componenti grafici

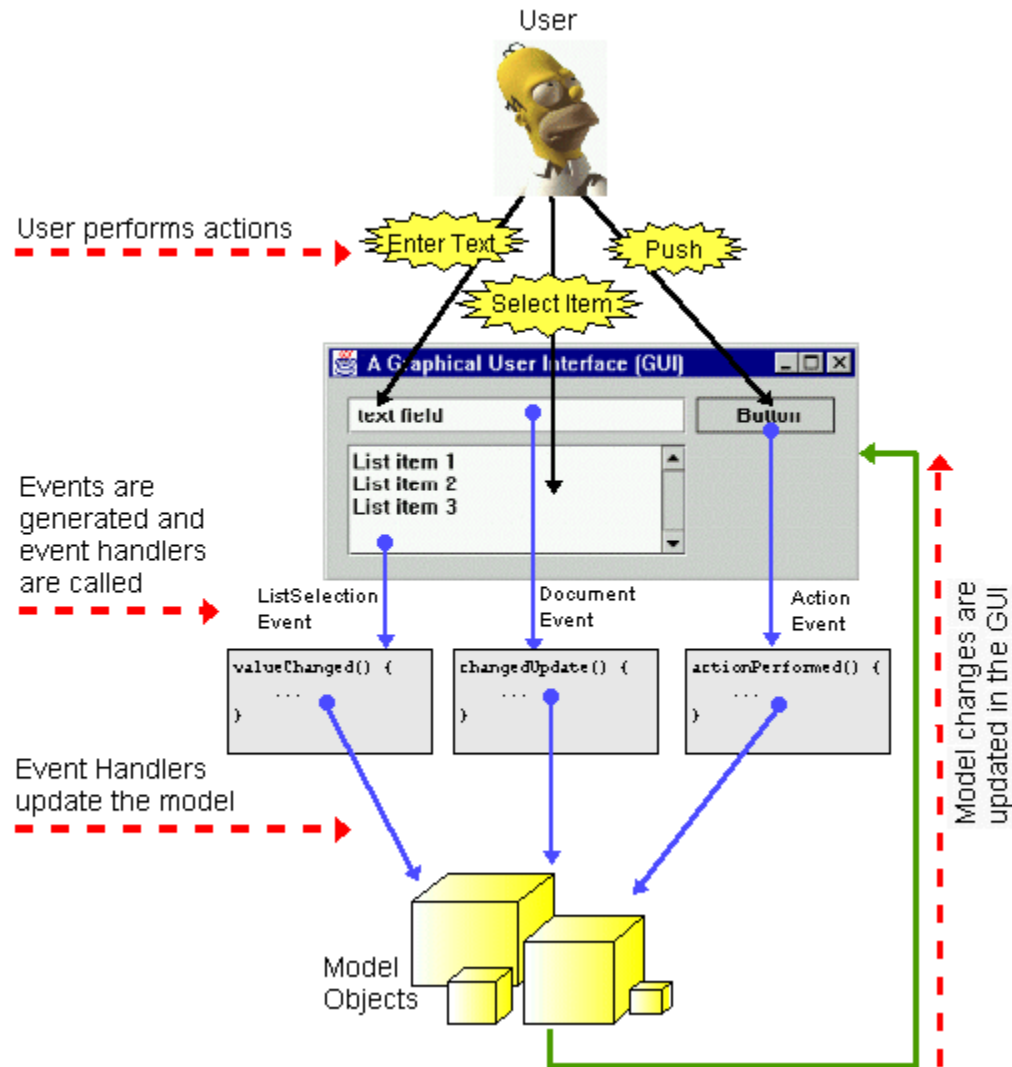


- ***model***
 - is made by the developer
- ***view***
 - set of objects of classes that inherit from `java.awt.component`
- ***controller***
 - is the event processing thread, which captures and propagates the event to the view and the model
 - classes that implement `EventListener` type interfaces (`ActionListener`, `MouseListener`, `WindowListener`, etc..)

- il pattern **Observer** (Osservatore) definisce una **dipendenza** uno a molti fra un soggetto **osservato** e vari oggetti «**osservatori**», in modo che se il soggetto osservato modifica il suo stato, a tutti gli oggetti osservatori che si sono esplicitamente registrati viene notificato il cambiamento avvenuto
- per poter notificare i cambiamenti di stato del soggetto agli osservatori, viene richiesto agli osservatori di **sottoscrivarsi** presso il soggetto
- il soggetto mantiene una **lista** degli **osservatori** registrati, per notificare a ciascuno di essi i propri cambiamenti di stato invocando un metodo specifico

- una classe astratta o una interfaccia definisce il ***prototipo del metodo*** di notifica
 - la sottoscrizione di un'istanza di un oggetto osservatore presso il soggetto rende disponibile un metodo specifico da invocare a ogni aggiornamento dello stato
- nella libreria Java AWT il pattern Observer viene utilizzato per realizzare gli ascoltatori (***Listener***) degli eventi asincroni generati dai componenti grafici

Elementi del pattern	Java AWT
Soggetto osservato	Componente <i>Button</i>
Osservatore astratto	Interfaccia <i>ActionListener</i>
Osservatore concreto	Oggetto che implementa l'interfaccia <i>ActionListener</i>



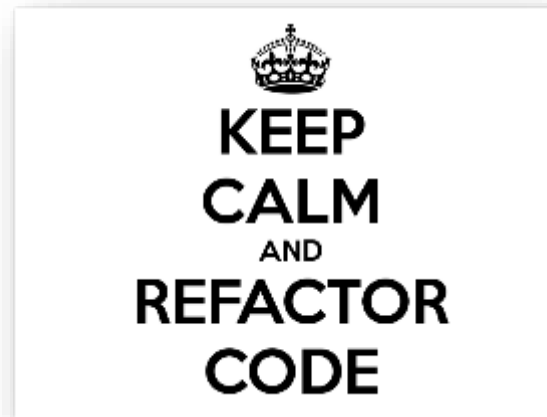
- gli anti-pattern sono dei design pattern che pur essendo utilizzabili, si rivelano successivamente inadatti o contro produttivi nella pratica
- il termine fu coniato nel 1995 da Andrew Koenig



an anti-pattern is a solution that initially looks like an attractive road lined with flowers, but further on leads you into a maze filled with monsters



- evitare *code smell*:
 - identificatori non significativi (i,j,k ...)
 - valori "cablati" nel codice (hard code)
 - copy and paste programming (Don't Repeat Yourself!)
- *anti-pattern*
 - *dead code* (codice irraggiungibile)
 - *spaghetti code* (flusso incomprensibile)
 - *kitchen sink* (lavello della cucina) o *swiss army knife* ("coltellino svizzero")
 - classe che contiene un gran numero di operazioni complesse ed eterogenee tra loro



code refactoring

- il ***code refactoring*** è una tecnica per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno
- l'obiettivo è ***migliorare*** alcune ***caratteristiche non funzionali*** del software:
 - ***leggibilità***
 - ***manutenibilità***
 - ***riusabilità***
 - ***estendibilità***
 - eliminazione di ***code smell***
- molti ambienti di sviluppo offrono valide funzionalità di ausilio al refactoring

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior
[Martin Fowler]