# Simulatore di Gestione della Memoria di un Elaboratore SiGeM

http://stylosoft.altervista.org
stylosoft@gmail.com

| Definizione del prodotto | 31 Marzo 2008 |
|---|---|
| Documento Esterno – Formale – v 2.0 | Definizione_del_prodotto_2.0.pdf |

## Redazione:

Giordano Cariani

## Revisione:

Daniele Bonaldo

## Approvazione:

Luca Rubin

## Lista di distribuzione:

Prof. Vardanega Tullio
Prof. Conte Renato
Prof. Palazzi Claudio
Stylosoft

## Registro delle modifiche:

| Versione | Data | Descrizione delle modifiche |
|---|---|---|
| 2.0 | 31/03/2008 | Inseriti diagrammi e aggiornato l'indice |
| 1.1 | 29/03/2008 | Correzioni varie e aggiunta diagrammi |
| 1.0 | 08/03/2008 | Completamento specifica tecnica e creazione appendice |
| 0.2 | 05/03/2008 | Creato scheletro e inserita specifica tecnica |
| 0.1 | 01/03/2008 | Prima stesura |

## Sommario

Questo documento ha il fine di illustrare lo standard seguito per lo sviluppo del sistema SiGeM.

# Indice

# 1 Introduzione

## 1.1 Scopo del documento

Tale documento descrive in modo dettagliato la specifica delle componenti relative allo sviluppo del progetto SiGeM.

## 1.2 Scopo del prodotto

Lo scopo del prodotto SiGeM è quello di fornire un sistema didattico destinato allo studio dei meccanismi di gestione della memoria in un elaboratore *multiprogrammato*. Tale software consentirà all'utilizzatore di simulare il comportamento di vari algoritmi di rimpiazzo delle *pagine* e *segmenti*, sulla base di dati da lui specificati, ai fini di illustrare le problematiche inerenti alla gestione della memoria. Per una descrizione più dettagliata rimandiamo al documento [AR].

## 1.3 Glossario

I termini in *corsivo* sono descritti per una migliore comprensione in [G], allegato a questo documento.

## 1.4 Riferimenti

- [AR] Analisi_dei_requisiti_2.0.pdf
- [SF] Studio_di_fattibilita_1.2.pdf
- [G] Glossario_3.0.pdf
- [PP] Piano_di_progetto_3.0.pdf
- [NP] Norme_di_progetto_3.0.pdf
- [NI] Norme_interne_3.0.pdf
- [ST] Specifica_tecnica_3.0.pdf

# 2 Standard di progetto

## 2.1 Standard di progettazione architetturale

Per la realizzazione dei diagrammi è stato seguito lo standard *UML* v2.0.

## 2.2 Standard di documentazione del codice

Per la documentazione del codice sono stati usati commenti nel formato Javadoc, in modo da poter poi generare la documentazione dei file sorgenti in maniera automatica.

## 2.3 Standard di programmazione

Come linguaggio di programmazione è stato deciso di usare il linguaggio di programmazione *Java* della Sun Microsystem Inc. versione 1.6.

Per maggiori dettagli si rimanda a [NP] al paragrafo “Codifica”.

## 2.4 Standard di denominazione di entità, relazione ed oggetti

Per la loro denominazione di entità è stata usata la convenzione di usare la prima lettera di ogni parola maiuscola e senza spazi. Per la denominazione delle relazioni tra le entità è stata usata la convenzione di mettere una parola o una frase significativa che espliciti il significato della relazione.

## 2.5 Strumenti di lavoro

Per la realizzazione dei diagrammi *UML* e per la programmazione è stato scelto di utilizzare l'ambiente di sviluppo NetBeans 6 (www.netbeans.org) in quanto distribuito con licenza GPL

# 3 Specifica delle componenti

## 3.1 Diagrammi delle classi

### 3.1.1 gui

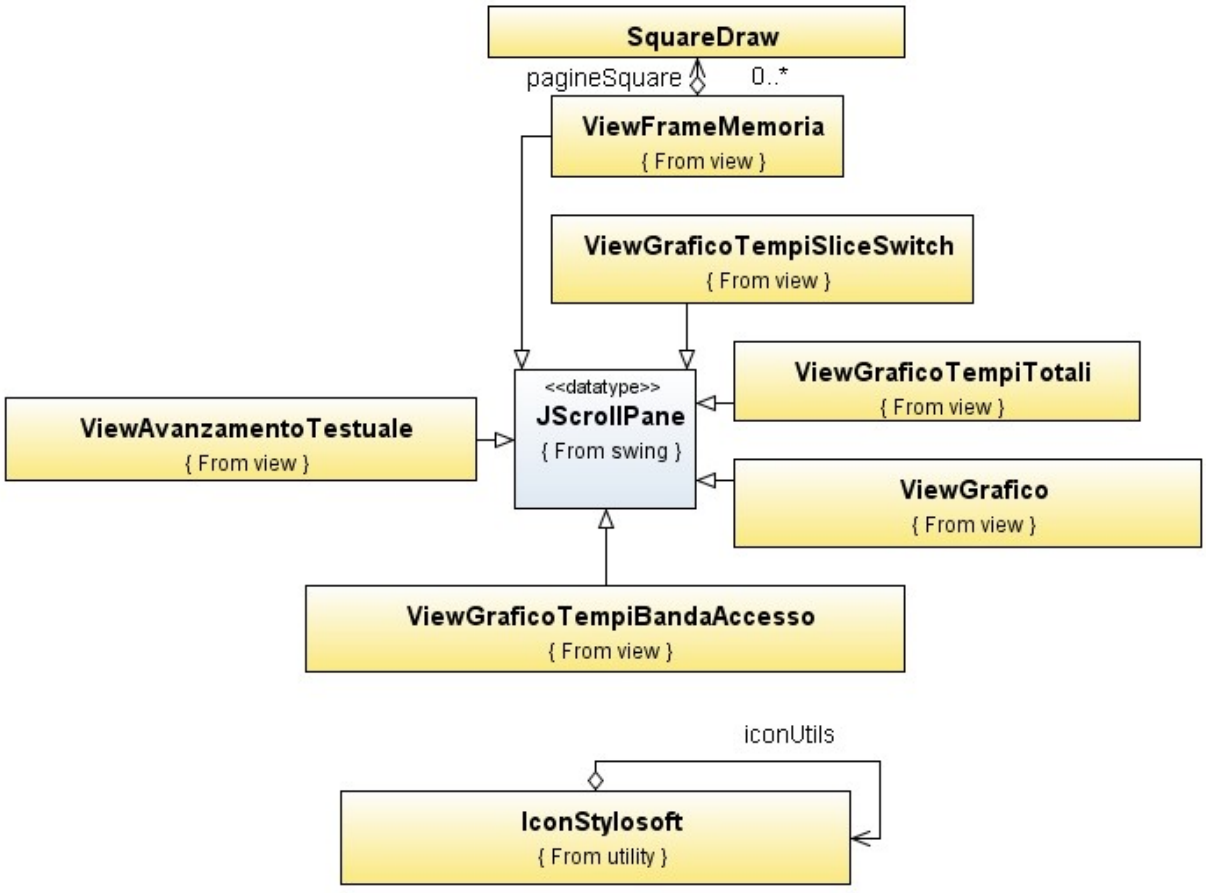



Figura 1

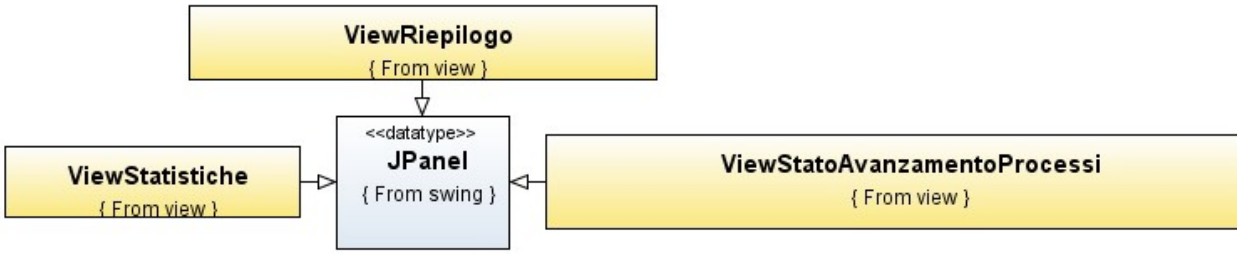
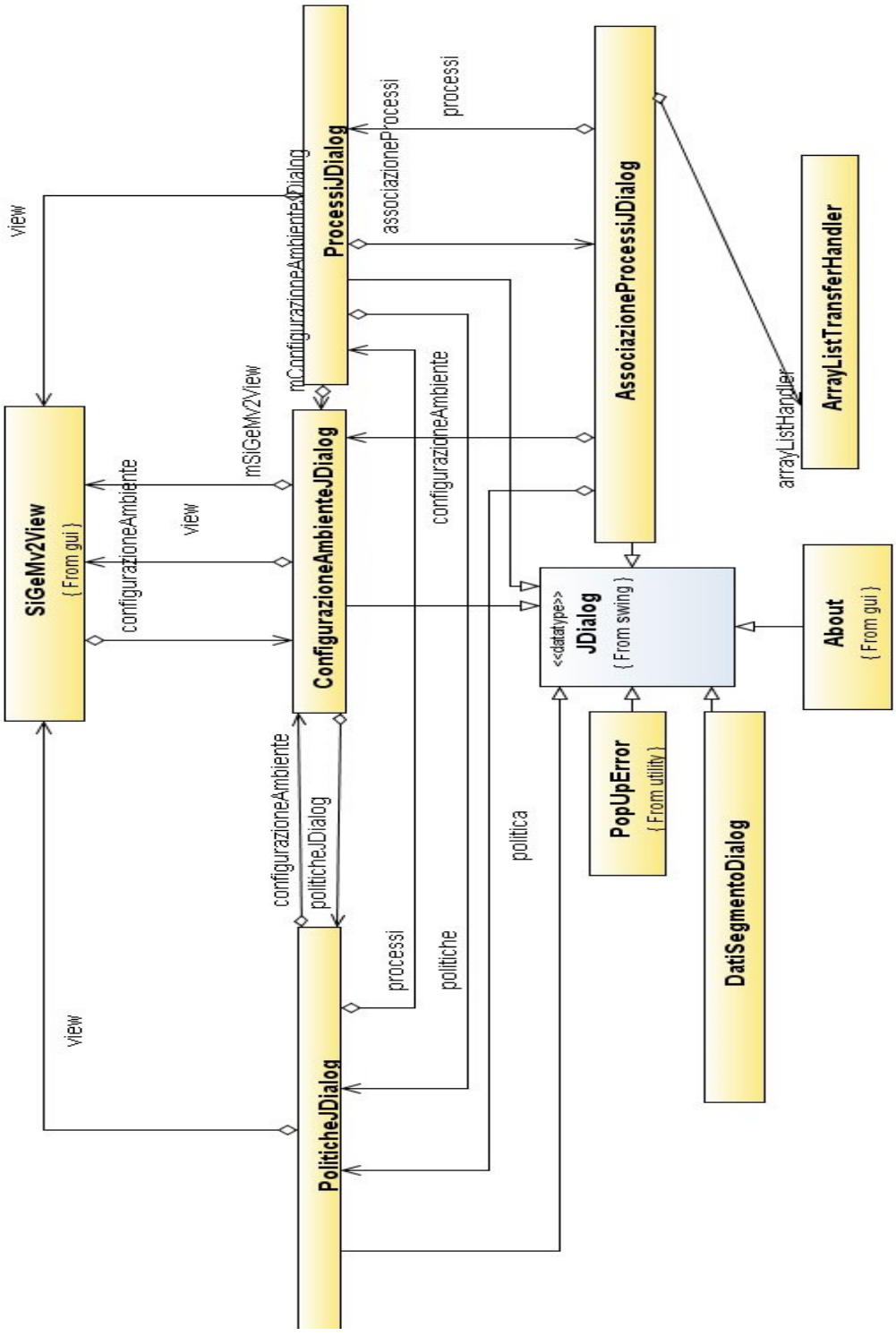
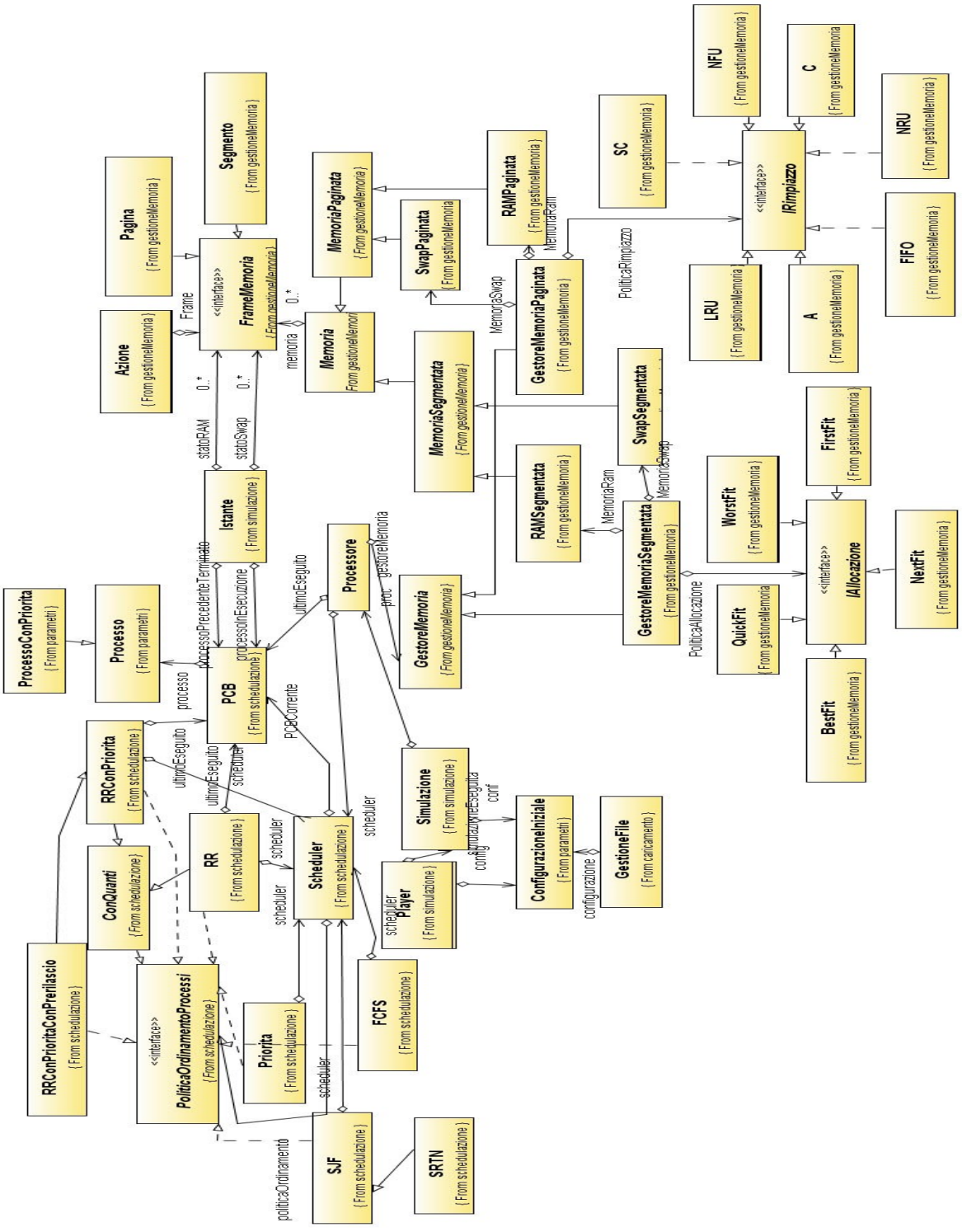

ViewRiepilogo
{ From view }
<<datatype>>
JPanel
{ From swing }
ViewStatistiche
{ From view }
ViewStatoAvanzamentoProcessi
{ From view }

Figura 2

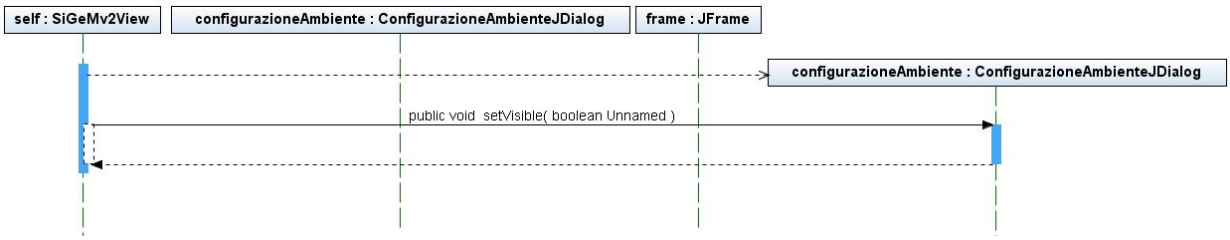

ProcessiJDialog
AssociazioneProcessiJDialog
ArrayListTransferHandler
SiGeMv2View
{ From gui }
ConfigurazioneAmbienteJDialog
<<datatype>>
JDialog
{ From swing }
About
{ From gui }
PopUpError
{ From utility }
DatiSegmentoDialog
PoliticheJDialog
view
processi
associazioneProcessi
mConfigurazioneAmbienteJDialog
mSiGeMv2View
configurazioneAmbiente
politicheJDialog
politiche
politica
arrayListHandler

### 3.1.2 logic

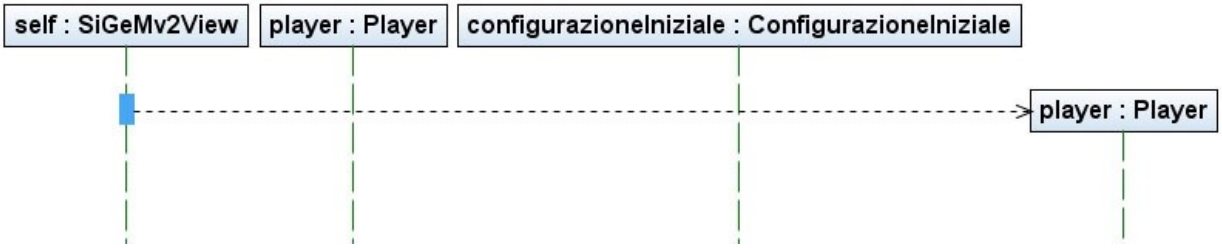

Segmento
{ From gestioneMemoria }
Pagina
{ From gestioneMemoria }
Azione
{ From gestioneMemoria }
Frame
«interface»
FrameMemoria
{ From gestioneMemoria }
MemoriaPaginata
{ From gestioneMemoria }
SwapPaginata
{ From gestioneMemoria }
RAMPaginata
{ From gestioneMemoria }
Memoria
{ From gestioneMemoria }
GestoreMemoriaPaginata
{ From gestioneMemoria }
MemoriaSegmentata
{ From gestioneMemoria }
SwapSegmentata
{ From gestioneMemoria }
RAMSegmentata
{ From gestioneMemoria }
GestoreMemoriaSegmentata
{ From gestioneMemoria }
GestoreMemoria
{ From gestioneMemoria }
Istante
{ From simulazione }
Processore
{ From simulazione }
ProcessoConPriorita
{ From parametri }
Processo
{ From parametri }
PCB
{ From schedulazione }
Simulazione
{ From simulazione }
Player
{ From simulazione }
ConfigurazioneIniziale
{ From parametri }
GestioneFile
{ From caricamento }
Scheduler
{ From schedulazione }
RR
{ From schedulazione }
RRConPriorita
{ From schedulazione }
ConQuanti
{ From schedulazione }
RRConPrioritaConPreriascio
{ From schedulazione }
«interface»
PoliticaOrdinamentoProcessi
{ From schedulazione }
Priorita
{ From schedulazione }
FCFS
{ From schedulazione }
SJF
{ From schedulazione }
SRTN
{ From schedulazione }
NFU
{ From gestioneMemoria }
SC
{ From gestioneMemoria }
C
{ From gestioneMemoria }
NRU
{ From gestioneMemoria }
«interface»
IRimpiazzo
LRU
{ From gestioneMemoria }
A
{ From gestioneMemoria }
FIFO
{ From gestioneMemoria }
FirstFit
{ From gestioneMemoria }
WorstFit
{ From gestioneMemoria }
NextFit
{ From gestioneMemoria }
QuickFit
{ From gestioneMemoria }
BestFit
{ From gestioneMemoria }
«interface»
IAllocazione
statoRAM
statoSwap
memoria
memoriaSwap
memoriaRam
PoliticaRimpiazzo
PoliticaAllocazione
processoPrecedenteTerminato
processoInEsecuzione
ultimoEseguito
processo
PCBCorrente
proc
gestoreMemoria
ultimoEseguito
scheduler
simulazioneEseguita
conf
configurazione
politicaOrdinamento

## 3.2 Diagrammi di sequenza

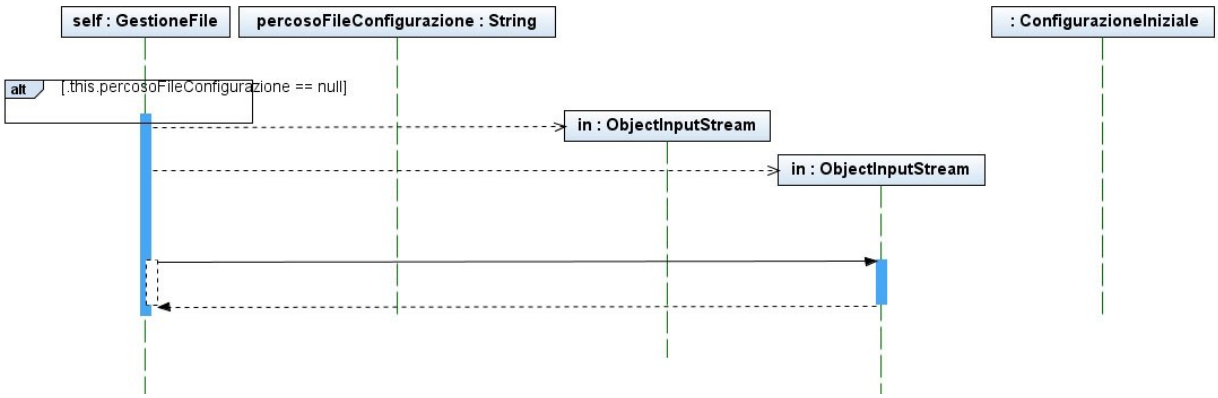
### 3.2.1 Creazione configurazione iniziale tramite wizard



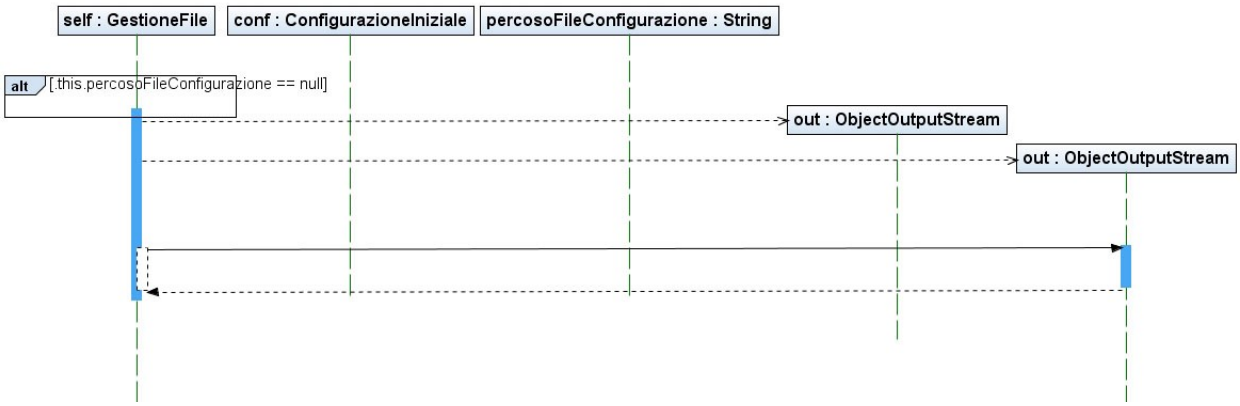

#### 3.2.1.1 Creazione player a partire da una configurazione qualsiasi



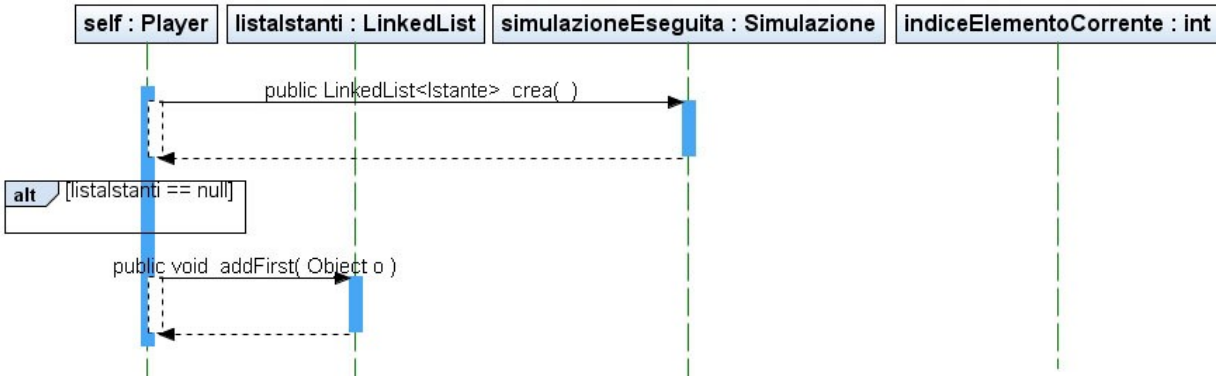

#### 3.2.1.2 Caricamento configurazione iniziale a partire da un file



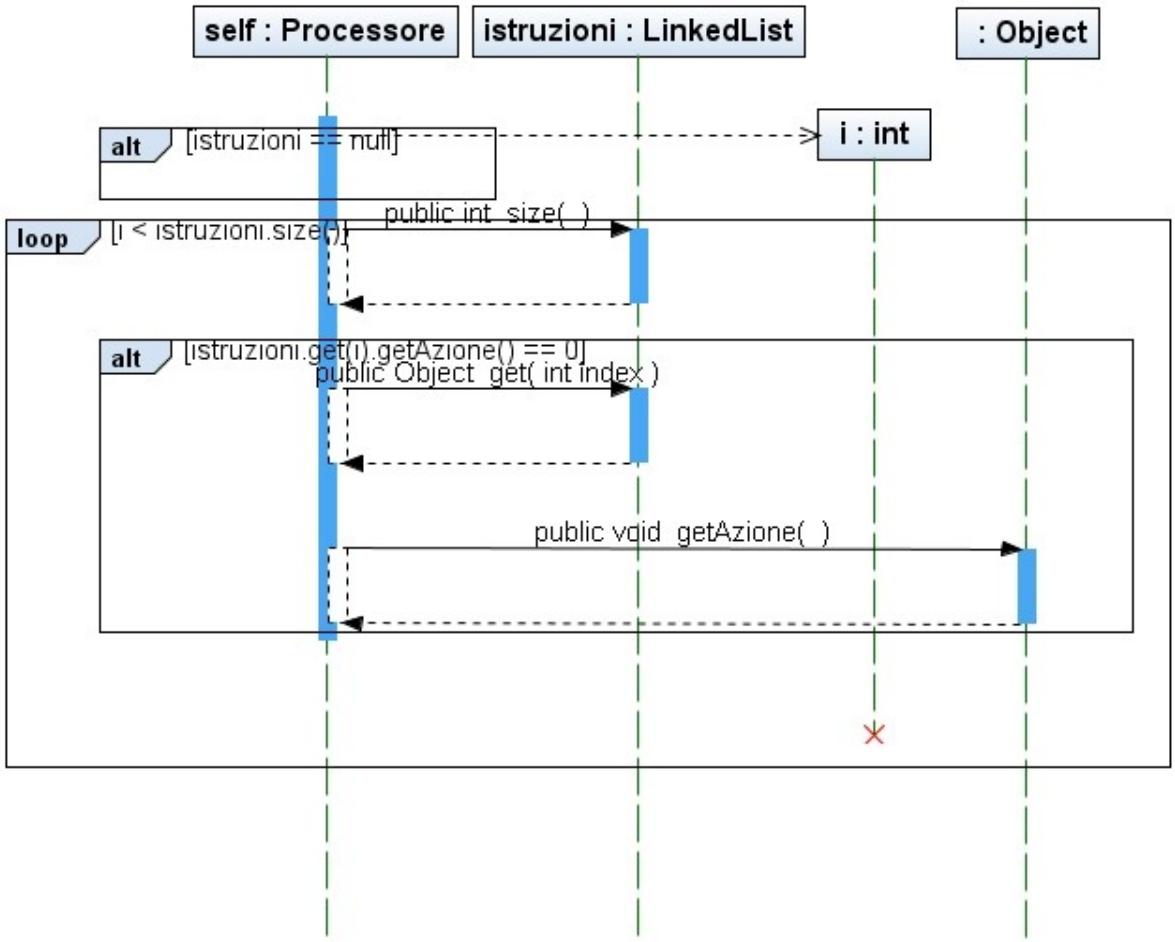
### 3.2.1.3 Salvataggio di una configurazione in un file



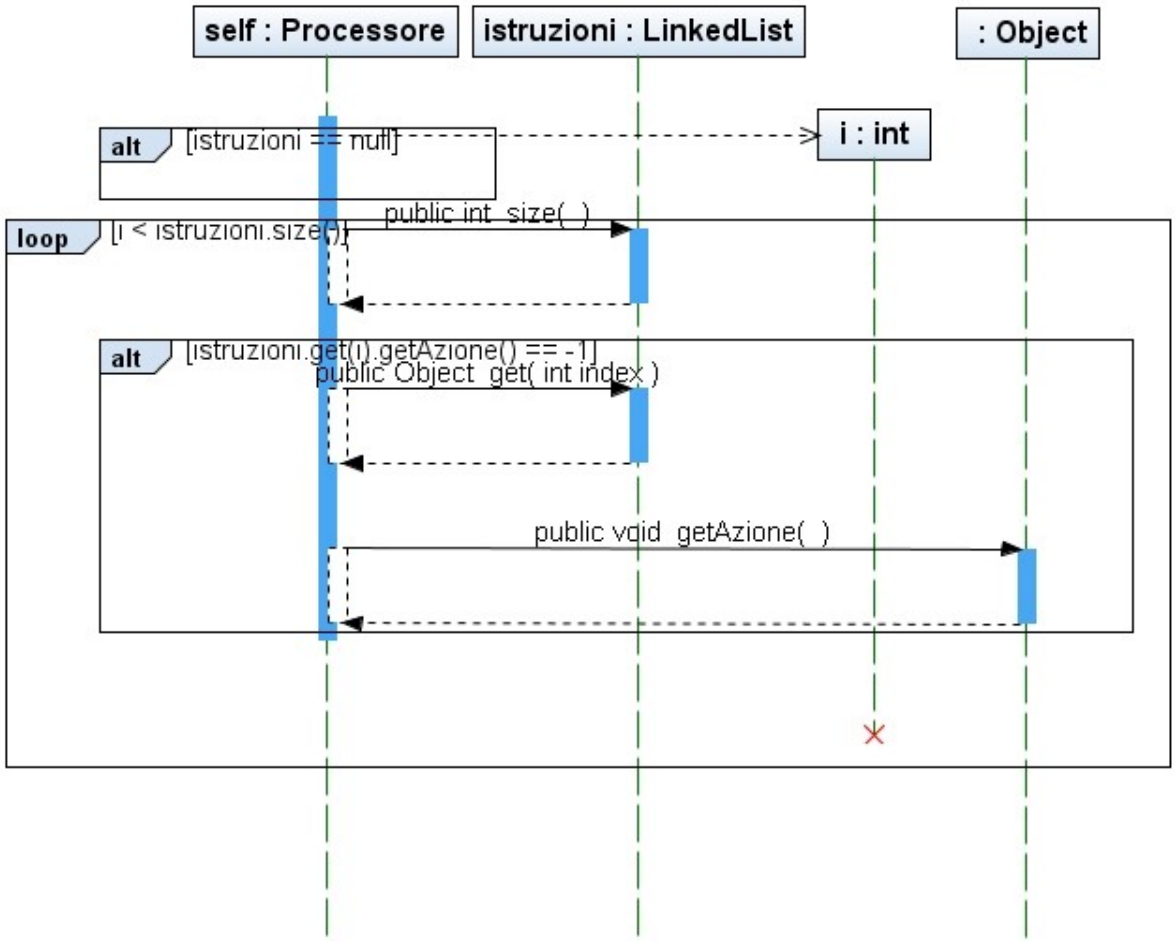

### 3.2.1.4 Creazione configurazione nel Player



### 3.2.1.5 Controllo riempimento RAM



#### 3.2.1.6 Controllo riempimento SWAP




## 3.3 Interfaccia grafica

### 3.3.1 Gui

#### 3.3.1.1 About

- Tipo, obiettivo e funzione della classe/interfaccia:
  E' la classe dove vengono mostrate tutte informazioni relative al fornitore del

prodotto SiGeM

- Relazioni d'uso con altre componenti:
  Nessuno

#### 3.3.1.2 SiGeMv2View

- Tipo, obiettivo e funzione della classe/interfaccia:
  E' la classe principale creata per far partire l'interfaccia grafica dell'applicazione SigeMv2.

- Relazioni d'uso con altre componenti:
  Utilizza i metodi getGeneralIcon della classe IconStylosoft

- Attività svolte e dati trattati:
  - Campi dati:
    - private RootWindow rootWindow
      La finestra principale alla quale vengono aggiunte le View, i menu e leToolBar.
    - private View[] views
      Array delle viste statiche
    - private ViewMap viewMap
      Contiene le viste statiche
    - private JMenuItem[] viewItems
      Gli elementi del menu view
    - private DockingWindowsTheme currentTheme
      Il tema applicato all'interfaccia grafica
    - private RootWindowProperties properties
      Un oggetto che contiene tutte le modifiche apportate alle caratteristiche di pulsanti, menù, grafica
    - private ActionListener lnrWindow
      Contiene la gestione degli eventi del menù Finestra
    - private JFrame frame

Il frame dell'applicazione

- private boolean statoGui
  Rappresenta lo stato della Gui, vale true se la simulazione e' in avanzamento automatico, false, se e' interrotta.

- private JButton jButtonSimulazionePlay
  Bottone per far partire la simulazione

- private JButton jButtonSimulazioneStop
  Bottone per stoppare la simulazione

- private JButton jButtonSimulazioneInizio
  Bottone per portare la simulazione allo stato iniziale

- private JButton jButtonSimulazioneFine
  Bottone per portare la simulazione allo stato finale

- private JButton jButtonSimulazioneIndietro
  Bottone per portare manualmente la simulazione indietro di un istante

- private JButton jButtonSimulazioneAvanti
  Bottone per portare manualmente la simulazione avanti di un istante

- private JButton jButtonSimulazionePausa
  Bottone per mettere in pausa la simulazione

- private JButton jButtonIndietroSignificativo
  Bottone per portare manualmente la simulazione indietro al primo istante scelto dall'utente (per esempio quando avviene il fault di memoria)

- private JButton jButtonAvantiSignificativo
  Bottone per portare manualmente la simulazione avanti al primo istante scelto dall'utente (per esempio quando avviene il fault di memoria)

- private JButton jButtonNuovaConfigurazione
  Bottone per aprire il wizard per la nuova configurazione

- private JButton jButtonApriConfigurazione
  Bottone per aprire la finestra per caricare il file di configurazione (se è stata salvata dall'utente)

- private JButton jButtonSalvaConfigurazione
  Bottone per salvare la configurazione attuale in un file

- private JButton jButtonModificaConfigurazione
  Bottone per modificare la configurazione attuale

- private JSpinner scegliVelocita
  L'utente può scegliere la velocità della simulazione

- private JComboBox comboBoxSignificativo
  è la combo degli eventi dove l'utente può scegliere quando vorrà spostare la simulazione al primo istante significativo dell'evento scelto

- private JMenuItem jSimulazioneItemPlay
  Voce del menù “simulazione” per far partire la simulazione

- private JMenuItem jSimulazioneItemStop
  Voce del menù “simulazione” per stoppare la simulazione

- private JMenuItem jSimulazioneItemInizio
  Voce del menù “simulazione” per portare la simulazione allo stato iniziale

- private JMenuItem jSimulazioneItemFine
  Voce del menù “simulazione” per portare la simulazione allo stato finale

- private JMenuItem jSimulazioneItemIndietro
  Voce del menù “simulazione” per portare la simulazione indietro di un istante

- private JMenuItem jSimulazioneItemAvanti
  Voce del menù “simulazione” per portare la simulazione avanti di un istante

- private JMenuItem jSimulazioneItemPausa

Voce del menù “simulazione” per mettere la simulazione in pausa

- private JMenu jSimulazioneItemIndietroSignificativo
  E' un sottomenù dove vengono indicati tutti gli istanti significativi portando la simulazione all'indietro fino all'istante significativo scelto dall'utente

- private JMenu jSimulazioneItemAvantiSignificativo
  E' un sottomenù dove vengono indicati tutti gli istanti significativi portando la simulazione in avanti fino all'istante significativo scelto dall'utente

- private JMenuItem itemAvantiFault
  Voce del menù “Istante significativo successivo” che porta la simulazione avanti fino al primo istante in cui avviene il fault di memoria

- private JMenuItem itemAvantiSwitch
  Voce del menù “Istante significativo successivo” che porta la simulazione avanti fino al primo istante in cui avviene il context switch

- private JMenuItem itemAvantiFullRAM
  Voce del menù “Istante significativo successivo” che porta la simulazione avanti fino al primo istante in cui avviene il riempimento totale della memoria

- private JMenuItem itemAvantiFullSwap
  Voce del menù “Istante significativo successivo” che porta la simulazione avanti fino al primo istante in cui avviene il riempimento totale dello swap

- private JMenuItem itemAvantiFineProc
  Voce del menù “Istante significativo successivo” che porta la simulazione avanti fino al primo istante in cui avviene la terminazione del processo in corso

- private JMenuItem itemAvantiNuovoProc
  Voce del menù “Istante significativo successivo” che porta la simulazione avanti fino al primo istante in cui avviene l'arrivo di un nuovo processo

- private JMenuItem itemIndietroFault
  Voce del menù “Istante significativo precedente” che porta la simulazione indietro fino al primo istante in cui avviene il fault di memoria

- private JMenuItem itemIndietroSwitch
  Voce del menù “Istante significativo precedente” che porta la simulazione indietro fino al primo istante in cui avviene il context switch

- private JMenuItem itemIndietroFullRAM
  Voce del menù “Istante significativo precedente” che porta la simulazione indietro fino al primo istante in cui avviene il riempimento totale della memoria

- private JMenuItem itemIndietroFullSwap
  Voce del menù “Istante significativo precedente” che porta la simulazione indietro fino al primo istante in cui avviene il riempimento totale dello swap

- private JMenuItem itemIndietroFineProc
  Voce del menù “Istante significativo precedente” che porta la simulazione indietro fino al primo istante in cui avviene la terminazione del processo in corso

- private JMenuItem itemIndietroNuovoProc
  Voce del menù “Istante significativo precedente” che porta la simulazione indietro fino al primo istante in cui avviene l'arrivo di un nuovo processo

- private JMenuItem jFileItemNuovaConfigurazione
  Voce del menù “File” per aprire il wizard per la nuova configurazione

- private JMenuItem jFileItemApriConfigurazione
  Voce del menù “File” per aprire la finestra per caricare il file di configurazione (se è stata salvata dall'utente

- private JMenuItem jFileItemSalvaConfigurazione
  Voce del menù “File” per salvare la configurazione attuale in un file

- private JMenuItem jFileItemSalvaConfigurazioneConNome
  Voce del menù “File” per salvare la configurazione attuale in un file specificando il nome del file

- private JMenuItem jFileItemModificaConfigurazione
  Voce del menù “File” per modificare la configurazione attuale

- private JMenuItem jFileItemDefaultConfigurazione
  Voce del menù “File” per caricare la configurazione di default impostata dal sistema

- private JMenuItem jFileItemEsci
  Voce del menù “File” per terminare l'applicazione

- private GestioneFile gestioneFile
  Riferisce un'istanza per la gestione completa di un file

- private ConfigurazioneAmbienteJDialog configurazioneAmbiente
  Riferisce un'istanza per il wizard di configurazione

- private ConfigurazioneIniziale configurazioneIniziale
  Riferisce un'istanza per la configurazione iniziale prima della simulazione

- private Player player
  Riferisce un'istanza di player per impostare la simulazione ed eseguirla

- private Istante istante
  Riferisce un'istanza di istante

- LinkedList<Processo> processiEseguiti
  Riferisce una lista di istanze di Processo

- private Thread auto
  Thread per aumentare l'interattivita' del utente durante l'avanzamento automatico

- private int velocita

E' la velocità di avanzamento nella modalità automatica

- private Vector<Vector<Integer>> processiUltimati
  E' la lista dei processi ultimati nei vari istanti

- private LinkedList<Boolean> contextSwitchs
  E' la lista dei context-switch avvenuti nella simulazione

- private static final Processo PROC_VUOTO
  E' un'istanza fittizia di processo

- Costruttori:
  - public SiGeMv2View()
    Unico costruttore della classe SiGeMv2View.
    Inizializza e imposta l'interfaccia grafica.

- Metodi:
  - public void abilitaTutto()
    Metodo che abilita tutte le funzioni una volta caricata la configurazione iniziale

  - private void aggiornaComandi()
    Metodo che abilita i comandi in base alle necessità

  - private void apriFileConfigurazione()
    Apertura di un file di configurazione e successivo caricamento della simulazione

  - private void azzeraSimulazioneTestuale(int, int)
    Metodo che azzera il frame della visualizzazione testuale

  - private void azzeraStatisticheSimulazione()
    Metodo che azzera le statistiche sulla vista ViewStatistiche

  - private boolean caricaSimulazioneDefault()
    Metodo che carica la configurazione di default

  - private void creaConfigurazione()
    Metodo che carica il primo passo del wizard di configurazione

- private void createRootWindow()
  Metodo che crea la root window e aggiunge le view statiche

- private void creaPlayer()
  Metodo che istanzia Player

- private JMenu createFileMenu()
  Metodo che crea tutte le voci del menù “File”

- private JMenu createHelpMenu()
  Metodo che crea tutte le voci del menù “Help”

- private JMenuBar createMenuBar() {
  Metodo che crea tutte le voci del menù e relative sottovoci

- private JMenu createSimulazioneMenu()
  Metodo che crea tutte le voci del menù “Simulazione”

- private JToolBar createToolBar()
  Metodo che crea tutti i bottoni

- private void defaultConfigurazione()
  Metodo che imposta la configurazione di default

- private void setDefaultLayout()
  Metodo che imposta il layout di default delle viste

- private void elaboraContextSwitch(LinkedList<Istante>)
  Metodo che elabora tutti i context switch

- private void elaboraProcessiTerminati(LinkedList<Istante>)
  Metodo che elabora tutti i processi terminati

- private void modificaConfigurazione()
  Procedura per la modifica della configurazione correntemente caricata

- private void salvaConfigurazione()
  Metodo che salva la configurazione in un file

- private void salvaConfigurazioneConNome()

Metodo che salva la configurazione in un file specificando un nome

- public void setIstanteZero()
  Metodo che inizializza le viste e prepara l'applicazione per l'esecuzione della simulazione

- private void simulazioneAvanti()
  Metodo che porta la simulazione all'istante successivo

- private boolean simulazioneCarica()
  Metodo che crea la simulazione

- private void simulazioneFine()
  Metodo che porta la simulazione all'istante finale

- private void showFrame()
  Metodo che inizializza il frame principale e lo visualizza

- private void simulazioneIndietro()
  Metodo che porta la simulazione all'istante precedente

- private void simulazioneInizio()
  Metodo che porta la simulazione all'inizio

- private void simulazionePausa()
  Metodo che stoppa temporaneamente la simulazione

- private synchronized void simulazionePlay()
  Avanzamento automatico della simulazione

- private void simulazioneSignificativoPrecedente(int scelta)
  Metodo che porta la simulazione all'istante significativo precedente

- private void simulazioneStop()
  Metodo che stoppa la simulazione portandola all'istante iniziale

- private void simulazioneSignificativoSuccessivo(int)
  Metodo che porta la simulazione all'istante significativo successivo

- private void updateViews(DockingWindow window, boolean added)

Metodo che aggiunge o rimuove viste dinamiche. Abilita o disabilita viste statiche

- private void visualizzaGrafico()
  Metodo che disegna la vista del grafico
- private void visualizzaGraficoTempi()
  Metodo che disegna la vista del grafico dei tempi
- public void visualizzaOrdProcessi(LinkedList<Processo>)
  Metodo che disegna la vista del grafico di avanzamento dei processi
- private void visualizzaRiepilogo()
  Metodo che disegna la vista del riepilogo
- private void visualizzaSimulazioneTestuale(Istante, int )
  Metodo che disegna la vista della simulazione testuale
- public void visualizzaStatistiche((Player player, Istante, boolean, int )
  Metodo che disegna la vista delle statistiche

### 3.3.2 gui.dialog

#### 3.3.2.1 ArrayListTransferHandler

- Tipo, obiettivo e funzione della classe/interfaccia:
  Classe che serve per gestire correttamente il drag&drop
- Relazioni d'uso con altre componenti:
  Nessuno
- **Interfacce e relazioni di uso da altre componenti:**
  Utilizzato dalla classe AssociazioniProcessiJDialog per gestire il drag&drop
- Attività svolte e dati trattati:
  - Campi dati:
    - int addCount
      Numero di items caricati

- int addIndex
  Posizione dove è stato aggiunto l'item

- int dimensioneRAM
  dimensione della RAM

- **Costruttori**:
  - public ArrayListTransferHandler(int dimensioneRAM)
    Unico costruttore che costruisce la classe a partire della dimensione della RAM espressa in KB

- Metodi:
  - public boolean canImport(JComponent c, DataFlavor[] flavors)
    Metodo che indica se un componente puo' accettare degli oggetti lasciati (dropped )dal Drag and Drop

  - protected Transferable createTransferable(JComponent c)
    Metodo che crea il componente da trasferire

  - protected void exportDone(JComponent c, Transferable data, int action)
    Metodo invocato dopo che l'operazione di spostamento e' stata eseguita

  - public int getSourceActions(JComponent c)
    Metodo che ritorna un intero rappresentante l'azione che il componente passato puo' eseguire

  - public boolean importData(JComponent c, Transferable t)
    Metodo chiamato quando si rilasciano degli oggetti su un componente. Chiama gli altri metodi per controllare se il componente puo' accettare gli oggetti passati ed e' incaricato di controllare che, nello specifico, la dimensione dei FrameMemoria spostati sommata a quella dei FrameMemoria gia' presenti, non superi la dimensione della RAM.

#### 3.3.2.2 AssociazioniProcessiJDialog

- Tipo, obiettivo e funzione della classe/interfaccia:
  E' il 4° passo della configurazione necessaria per far partire correttamente la simulazione

- Relazioni d'uso con altre componenti:

Viene chiamato da ProcessiJDialog (3° passo della configurazione)

- Interfacce e relazioni di uso da altre componenti:
  Nessuna (vengono utilizzate le sue implementazioni).

- Attività svolte e dati trattati:
  - Campi dati:
    - private ArrayListTransferHandler arrayListHandler
      campo incaricato di gestire il Drag&Drop

    - private ConfigurazioneIniziale confIniziale
      Configurazione iniziale preesistente in caso di modifica configurazione

    - private ConfigurazioneAmbienteJDialog configurazioneAmbiente
      campo dove contengono tutte le informazioni relative al 1° passo

    - private PoliticheJDialog politica;
      campo dove contengono tutte le informazioni relative al 2° passo

    - private ProcessiJDialog processi;
      campo dove contengono tutte le informazioni relative al 3° passo

    - private SiGeMv2View view;
      campo dove contengono tutte le informazioni relative al frame

    - private ConfigurazioneIniziale confIniziale;
      campo per inizializzare la configurazione iniziale

    - private LinkedList<Processo> listaProcessi
      campo dove vengono caricati tutti i processi impostati dall'utente

  - Costruttori:
    - public AssociazioneProcessiJDialog(java.awt.Frame parent, boolean modal, ConfigurazioneAmbienteJDialog configurazione, PoliticheJDialog pol, ProcessiJDialog proc, SiGeMv2View view)
      1° costruttore: imposta l'interfaccia grafica del 4° passo del wizard di configurazione.

    - public AssociazioneProcessiJDialog(java.awt.Frame parent, boolean modal, ConfigurazioneAmbienteJDialog configurazione, PoliticheJDialog pol,

ProcessiJDialog proc, SiGeMv2View view, ConfigurazioneIniziale confIniziale)
2° costruttore: imposta il 4° passo per la modifica della configurazione

- Metodi:
  - private boolean accessoValido(FrameMemoria frame, int indiceLista)
    Metodo che controlla la validità di un accesso al suo caricamento, controllando se lo spazio disponibile in RAM e' sufficiente a contenere il FrameMemoria, considerando i FrameMemoria già caricati
  - private void aggiornaListaFrame()
    Metodo che aggiorna il modello per la lista dei FrameMemoria di un processo quando viene selezionato un'altro processo nel JTabbedPane. E' necessario perché ogni processo ha i propri FrameMemoria
  - private void caricaAccessi()
    Metodo che carica gli accessi dalla ConfigurazioneIniziale che si sta modificando
  - private void creaAccessi()
    Metodo che crea ed associa gli Accessi ai vari FrameMemoria ai processi
  - private void creaMenu()
    Metodo incaricato di creare il menu che apparira' nelle varie JList con il click destro
  - private JScrollPane creaPannelloProcesso(int IdProcesso)
    Metodo che crea un JScrollPane contenente tutte le liste rappresentanti gli istanti di esecuzione del processo passato per parametro. Ogni lista e' contenuta in un JScrollPane indipendente in modo da poter effettuare lo scrolling se ci sono troppi FrameMemoria
  - private int getNuovoIndirizzo(int idProcesso)
    Metodo che Ritorna il primo intero rappresentante l'indirizzo del primo frameMemoria disponibile per il processo indicato
  - private void initComponents()

Metodo che carica tutte la grafica relativa al wizard con inizializzano della tabella dei processi

- private void inizializzaConfigurazioneIniziale()
  Metodo che imposta la configurazione quando l'utente ha completato di impostare tutti i parametri necessari per la simulazione

- private void jButtonAnnullaActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che definisce l'evento di annullamento della configurazione della simulazione.

- private void jButtonEliminaActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che cancella le pagine o i segmenti caricati nella lista

- private void jButtonHelpActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che porta al link dell'help in html

- private void jButtonModificaActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che permette di modificare la dimensione del segmento/pagina

- private void jButtonNuovoFrameActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che permette di aggiungere nuove pagine o segmenti da associare ai processi

- private void jButtonIndietroActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che definisce l'evento di ritorno al 3° passo del wizard di configurazione

- private void jTabbedPaneProcessiStateChanged(javax.swing.event.ChangeEvent evt)
  Metodo che aggiorna la lista frame

- private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che definisce l'evento del completamento del wizard.

#### 3.3.2.3 ConfigurazioneAmbienteJDialog

- **Tipo, obiettivo e funzione della classe/interfaccia:**

Classe che rappresenta il 1° del wizard per la configurazione della simulazione

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene richiamata da SigeMv2View per la configurazione della simulazione

- **Attività svolte e dati trattati:**
  - Campi dati:
    - private static final int DIMPROCESSI
      campo di dimensione max per impostare l'intervallo di valori del numero di processi da impostare
    - private static final int DIMRAM
      campo di dimensione max per impostare l'intervallo di valori della dimensione della RAM
    - private static final int DIMSWAP
      campo di dimensione max per impostare l'intervallo di valori della dimensione dello SWAP
    - private static final int DIMPAGINE
      campo di dimensione max per impostare l'intervallo di valori della dimensione della pagina
    - private static final int DIMBUS
      campo di dimensione max per impostare l'intervallo di valori della dimensione della banda bus
    - private int numProcessi;
      campo dove verrà salvato il numero di processi impostati dall'utente
    - private int dimensioneRAM;
      campo dove verrà salvato la dimensione della RAM impostata dall'utente
    - private int dimensioneAreaSWAP;
      campo dove verrà salvato la dimensione dello SWAP impostata dall'utente

- private int dimensionePagina;
  campo dove verrà salvato la dimensione della pagina impostata dall'utente

- private int tempoContextSwitch;
  campo dove verrà salvato il tempo di context switch impostata dall'utente

- private int tempoAccessoDisco;
  campo dove verrà salvato il tempo di accesso al disco impostato dall'utente

- private int bandaBusDati;
  campo dove verrà salvato la dimensione della banda di bus di dati impostata dall'utente

- private SiGeMv2View view
  campo dove contengono tutte le informazioni relative al frame

- Costruttori:
  - public ConfigurazioneAmbienteJDialog(java.awt.Frame parent, boolean modal, SiGeMv2View view)
    costruttore della classe che imposta l'interfaccia grafica del 1° passo del wizard di configurazione della simulazione

  - public ConfigurazioneAmbienteJDialog(java.awt.Frame parent, boolean modal, SiGeMv2View view, ConfigurazioneIniziale confIniziale)
    costruttore delle classe che imposta l'interfaccia grafica del 1° passo del wizard per la modifica della configurazione attuale

- **Metodi**:
  - private boolean controlliLogici()
    Metodo che controlla se i valori impostati sono corretti o meno

  - private Integer[] impostaBUS()
    imposta l'intervallo di valori per la banda di bus dati

  - private Integer[] impostaDimensioniRAM()
    imposta l'intervallo di valori per la dimensione della RAM

- private Integer[] impostaDimensioniSWAP()
  imposta l'intervallo di valori per la dimensione dell'area SWAP

- private Integer[] impostaNumProcessi()
  imposta l'intervallo di valori per il numero di processi

- private Integer[] impostaPagine()
  imposta l'intervallo di valori per la dimensione delle pagine

- private Integer[] impostaTempi()
  imposta l'intervallo dei valori di tempo

- private void initComponents()
  inizializza e caricare l'interfaccia grafica del 1° passo del wizard

- private void initJSpinnerProcessi()
  imposta lo spinner con l'intervallo dei valori del numero di processi

- private void initSpinnerBandaBusDati()
  imposta lo spinner con l'intervallo dei valori della banda bus di dati

- private void initSpinnerDimensioneRAM()
  imposta lo spinner con l'intervallo dei valori della dim. della RAM

- private void initSpinnerDimensioneSWAP()
  imposta lo spinner con l'intervallo dei valori della dim. dello SWAP

- private void initSpinnerPagine()
  imposta lo spinner con l'intervallo dei valori della dim. delle pagine

- private void initJSpinnerProcessi()
  imposta lo spinner con l'intervallo dei valori del numero di processi

- private void initSpinnerTempi()
  imposta lo spinner con l'intervallo dei valori di tempo per il context switch e per il tempo di accesso al disco

- private void jButtonAnnullaActionPerformed(java.awt.event.ActionEvent evt)
  metodo che gestisce l'annullamento della configurazione della simulazione

- private void jButtonHelpActionPerformed(java.awt.event.ActionEvent evt)
  metodo che porta al help in html
- private void jButtonAvantiActionPerformed(java.awt.event.ActionEvent evt)
  metodo che gestisce il caricamento del 2° passo del wizard e salva i valori impostati nel 1° passo

#### 3.3.2.4 DatiSegmentoDialog

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe permette di definire la dimensione del segmento scelto

- **Relazioni d'uso con altre componenti:**
  Nessuna

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dall'AssociazioneProcessiJDialog (4° passo del wizard) per definire la dimensione del segmento scelto

- **Attività svolte e dati trattati:**
  - Campi dati:
    - public static final int RET_CANCEL = 0
      A return status code - returned if Cancel button has been pressed
    - public static final int RET_OK = 1
      A return status code - returned if OK button has been pressed
    - private int returnStatus
      campo dove viene valorizzato lo status code
  - Costruttori:
    - public DatiSegmentoDialog(java.awt.Frame parent, boolean modal, int massimaDimensione)
      Costruttore che imposta l'interfaccia grafica
    - public DatiSegmentoDialog(java.awt.Frame parent, boolean modal, int massimaDimensione, int dimensioneAttuale)
      Costruttore che imposta l'interfaccia per la modifica della dimensione
  - **Metodi:**

- private void cancelButtonActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che chiude l'interfaccia

- private void closeDialog(java.awt.event.WindowEvent evt)
  Metodo che chiude la finestra

- private void initComponents()
  Metodo che imposta l'interfaccia grafica

- private void jSliderDimensioneStateChanged(javax.swing.event.ChangeEvent evt)
  Metodo che imposta la dimensione del segmento

- private void okButtonActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che prende i valori della dimensione del segmento e chiude l'interfaccia

#### 3.3.2.5 ModelloProcessi

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe imposta la tabella dei processi da configurare

- **Relazioni d'uso con altre componenti:**
  Nessuna

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da ProcessiJDialog per l'inizializzazione della tabella del 3° passo del wizard

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private String[] nomiColonna
      Array dove contengono i titoli delle colonne

    - private Object[][] contenutiRighe
      Matrice con i valori delle celle

  - **Costruttori**:
    - public ModelloProcessi(int nRighe)

Unico costruttore che si occupa dell'inizializzazione delle celle della tabella dell'associzione

- public ModelloProcessi(LinkedList<Processo> processi, int nRighe)
  costruttore che imposta la tabella con i dati da modificare (configurazione attuale)

- **Metodi**:
  - public void setValueAt(Object value, int row, int col)
    Metodo che ritorna il valore della cella scelta

### 3.3.2.6 ModelloProcessiPriorita

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe imposta la tabella dei processi da configurare

- **Relazioni d'uso con altre componenti:**
  Nessuna

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da ProcessiJDialog per l'inizializzazione della tabella del 3° passo del wizard

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private String[] nomiColonna
      Array dove contengono i titoli delle colonne

    - private Object[][] contenutiRighe
      Matrice con i valori delle celle

  - **Costruttori:**
    - public ModelloProcessiPriorita(int nRighe)
      Costruttore che si occupa dell'inizializzazione delle celle della tabella dell'associzione

    - public ModelloProcessiPriorita(LinkedList<Processo> processi, int nRighe)
      Costruttore che si occupa dell'inizializzazione delle celle della tabella dell'associzione per modificare poi la configurazione attuale

- **Metodi:**
  - public void setValueAt(Object value, int row, int col)
    Metodo che ritorna il valore della cella scelta

#### 3.3.2.7 PoliticheJDialog

- Tipo, obiettivo e funzione della classe/interfaccia:
  Questa classe rappresenta il 2° passo del wizard dove vengono scelte tutte le politiche necessarie per la simulazione

- Relazioni d'uso con altre componenti:
  Viene chiamato da ConfigurazioneAmbienteJDialog

- Interfacce e relazioni di uso da altre componenti:
  Nessuna

- Attività svolte e dati trattati:
  - Campi dati:
    - private ConfigurazioneIniziale confIniziale
      campo dove è salvata la configurazione iniziale da modificare
    - private ConfigurazioneAmbienteJDialog configurazioneAmbiente;
      campo dove contengono tutte le informazioni del 1° passo del wizard
    - private ProcessiJDialog processi;
      campo per l'inizializzazione del 3° passo del wizard
    - private SiGeMv2View view;
      campo dove contengono tutte le informazioni del frame
    - private int gestioneMemoria;
      campo dove verrà salvata l'informazione del tipo di gestione di memoria scelto dall'utente
    - private int politica;
      campo dove verrà salvata l'informazione della politica scelta dall'utente
    - private int politicaSchedulazione
      campo dove verrà salvata l'informazione della politica di schedulazione scelta dall'utente

- boolean modifica = false
  campo che permette di capire se si sta modificando la configurazione o no
- private int timeSlice
  campo dove viene salvato il valore del time slice

- Costruttori:
  - public PoliticheJDialog(java.awt.Frame parent, boolean modal, ConfigurazioneAmbienteJDialog conf, SiGeMv2View view)
    Costruttore che si occupa di inizializzare l'interfaccia grafica e di impostare i valori di default

  - public PoliticheJDialog(java.awt.Frame parent, boolean modal, ConfigurazioneAmbienteJDialog conf, SiGeMv2View view, ConfigurazioneIniziale confIniziale)
    Costruttore che si occupa di inizializzare l'interfaccia grafica e di impostare i valori della configurazione da modificare

- Metodi:
  - private void gestionePoliticaPagine(String politica)
    metodo che gestisce il salvataggio della politica della pagine

  - private void gestionePoliticaSegmenti(String politica)
    metodo che gestisce il salvataggio della politica dei segmenti

  - private void gestioneSchedulazione(String politica)
    metodo che gestisce il salvataggio della politica di schedulazione

  - private void initComponents()
    Metodo che inizializza l'interfaccia grafica e imposta i valori di default

  - private void jButtonAnnullaActionPerformed(java.awt.event.ActionEvent evt)
    Metodo che gestisce l'evento di annullamento della configurazione

  - private void jButtonAvantiActionPerformed(java.awt.event.ActionEvent evt)
    Metodo che gestisce l'evento di salvataggio delle informazioni relative al 2° passo e carica l'istanza di ProcessiJDialog per il 3° passo

  - private void jButtonHelpActionPerformed(java.awt.event.ActionEvent evt)
    Metodo che gestisce il collegamento alla pagina dell'help in html

- private void jButtonIndietroActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che gestisce il ritorno al primo passo

- private void jComboBoxRimpiazzoPagineActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che gestisce l'evento di cambio valore della combo box della politica delle pagine

- private void jComboBoxSchedulazioneActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che gestisce l'evento di cambio valore della combo box della politica di schedulazione

- private void jComboBoxSuddivisioneMemoriaActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che gestisce l'evento di cambio valore della combo box della suddivisione di memoria

#### 3.3.2.8 ProcessiJDialog

- Tipo, obiettivo e funzione della classe/interfaccia:
  Questa classe rappresenta il 3° passo del wizard dove vengono impostati tutti i processi

- Relazioni d'uso con altre componenti:
  Viene chiamato da PoliticheJDialog

- Interfacce e relazioni di uso da altre componenti:
  Nessuna

- Attività svolte e dati trattati:
  - Campi dati:
    - private ConfigurazioneAmbienteJDialog configurazioneAmbiente;
      campo dove contengono tutte le informazioni relative al 1° passo del wizard di configurazione

    - private PoliticheJDialog politiche

campo dove contengono tutte le informazioni relative al 2° passo del wizard di configurazione

- private AssociazioneProcessiJDialog associazioneProcessi
  campo che serve per istanzaire il 4° passo del wizard
- private SiGeMv2View view;
  campo dove contengono tutte le informazioni relative al frame
- private Object[][] combinazioneProcessi;
  campo matrice dove verranno salvati tutti i processi
- private ConfigurazioneIniziale confIniziale
  campo dove ci sono tutte le informazioni della configurazione iniziale da modificare
- boolean modifica = false
  campo che permette di capire se stiamo modificando la configurazione o no

- Costruttori:
  - public ProcessiJDialog(java.awt.Frame parent, boolean modal, ConfigurazioneAmbienteJDialog conf, PoliticheJDialog pol, SiGeMv2View view)
    Costruttore che imposta l'interfaccia grafica e imposta i valori di default
  - public ProcessiJDialog(java.awt.Frame parent, boolean modal, ConfigurazioneAmbienteJDialog conf, PoliticheJDialog pol, SiGeMv2View view, ConfigurazioneIniziale confIniziale)
    Costruttore che imposta l'interfaccia grafica per la modifica della configurazione attuale
- Metodi:
  - private void initComponents()
    Metodo che carica tutte la grafica relativa al wizard con inizializzano della tabella dei processi
  - private void initJTable()
    Metodo che imposta la tabella dei processi

- private void jButtonAnnullaActionPerformed(java.awt.event.ActionEvent evt)
  Metodo che gestisce l'annullamento della configurazione
- private void jButtonAvantiActionPerformed(java.awt.event.ActionEvent evt)

  Metodo che gestisce il salvataggio dei dati realative al 3° passo e carica il 4°.
- private void jButtonIndietroActionPerformed(java.awt.event.ActionEvent evt)

  Metodo che torna al 2° passo del wizard
- private void jButtonHelpActionPerformed(java.awt.event.ActionEvent evt)

  Metodo che gestisce il collegamento all'help in hmtl

### 3.3.3 gui.utility

#### 3.3.3.1 HelpHtml

- Tipo, obiettivo e funzione della classe/interfaccia:
  Classe che si occupa di gestire tutte le icone dell'interfaccia grafica.
- Relazioni d'uso con altre componenti:
  Nessuna.
- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per caricare le icone dei bottoni e delle view statiche
- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno
  - Costruttori:
    - Nessuno
  - Metodi:
    - public static void openUrl(String url)
      Metodo che gestisce il collegamento all'help specificando la pagina htm

da visualizzare

#### 3.3.3.2 IconStylosoft

- Tipo, obiettivo e funzione della classe/interfaccia:
  Classe che si occupa di gestire tutte le icone dell'interfaccia grafica.

- Relazioni d'uso con altre componenti:
  Nessuna.

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per caricare le icone dei bottoni e delle view statiche

- Attività svolte e dati trattati:
  - Campi dati:
    - private static IconStylosoft iconUtils
      Campo dati statico che carica l'immagine dell'icona

  - Costruttori:
    - Nessuno

  - Metodi:
    - public static ImageIcon getGeneralIcon(String name)
      Metodo che si occupa di recuperare l'icona

    - public ImageIcon getIcon(String name)
      Metodo che si occupa di recuperare l'icona tramite il nome dell'icona

#### 3.3.3.3 ImageFilter

- Tipo, obiettivo e funzione della classe/interfaccia:
  Classe che serve filtrare tutti i file con l'estensione sigem quando si sceglie il file da caricare

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:

Viene usata da SigeMv2View quando si apre il dialog per il caricamento del file di configurazione

- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno
  - Costruttori:
    - Nessuno
  - **Metodi**:
    - public boolean accept(File f)
      Metodo che accetta il file con estensione sigem
    - public String getDescription()
      Metodo che ritorna la descrizione del file da caricare

#### 3.3.3.4 PopUpError

**Tipo, obiettivo e funzione della classe/interfaccia:**

Classe che visualizza la descrizione dell'errore

- **Relazioni d'uso con altre componenti:**
  Nessuna
- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dal ConfigurazioneAmbienteJDialog quando l'utente sbaglia a impostare i valori del 1° passo del wizard
- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - Nessuno
  - **Costruttori:**
    - public PopUpError(java.awt.Frame parent, boolean modal, String messaggio)
      Unico costruttore che imposta l'interfaccia con il messaggio da visualizzare

- **Metodi**:
  - private void initComponents()
    Metodo che costruisce l'interfaccia grafica

  - private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt)
    Metodo che chiude l'interfaccia

#### 3.3.3.5 SquareDraw

- Tipo, obiettivo e funzione della classe/interfaccia:
  Classe che serve per disegnare le pagine della memoria

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da ViewFrameMemoria per disegnare i quadrati della memoria

- Attività svolte e dati trattati:
  - Campi dati:
    - Shape square
      campo dove vengono creati i quadrati delle pagine

    - Color color;
      campo dove memorizzare il colore del quadrato creato

    - int altezza
      campo dove viene salvata l'altezza della figura da disegnare

    - private String indirizzoFrame
      campo dove viene salvato il numero del processo che utilizza questa figura

    - private int xCoord
      campo dove viene salvata l'informazione della posizione nell'asse orizzontale

    - private int xCoord

campo dove viene salvata l'informazione della posizione nell'asse verticale

- Costruttori:
  - public SquareDraw(int xCoord, int yCoord, int lunghezza, int altezza, Color color, String text)
    Unico costruttore dove vengono definiti, per ogni figura, l'altezza, la lunghezza e la posizione del quadrato all'interno della finestra e relativa descrizione

### 3.3.4 gui.view

#### 3.3.4.1 ViewAvanzamentoTestuale

- Tipo, obiettivo e funzione della classe/interfaccia:
  Classe che permette la visualizzazione testuale dell'avanzamento della simulazione

- Relazioni d'uso con altre componenti:
  Nessuno

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del frame di di avanzamento testuale

- Attività svolte e dati trattati:
  - Campi dati:
    - private int istantiTotali;
      Numero di istanti totali della simulazione

    - private JTextArea testo
      Array che tutte le pagine create.

    - private int tipoGestioneMemoria
      Tipo di gestione della memoria

  - Costruttori:
    - public ViewAvanzamentoTestuale()

Unico costruttore che crea il pannello del frame

- **Metodi**:
  - public void aggiorna(Istante istante, int idIstanteCorrente)
    Metodo che aggiorna la tabella ad ogni istante
  - private void disegnaPagine(Graphics g)
    Metodo che si occupa di disegnare le pagine nel pannello
  - public void configura(int istantiTotali, int tipoGestioneMemoria)
    Metodo che inizializza il frame

#### 3.3.4.2 ViewFrameMemoria

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe estende JScrollPanel e crea la finestra delle statistiche
- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica della memoria
- Attività svolte e dati trattati:
  - Campi dati:
    - private static final int ALTEZZA=550
      Imposta l'altezza totale della RAM in caso di segmentazione
    - private static final int LATO=48
      Setta la grandezza dei quadrati rappresentanti le pagine
    - private int dimMemoria
      Dimensione totale della memoria in byte
    - private int numPagine
      Numero di pagine: utile per la visualizzazione iniziale di tutta la RAM. Rimane a 0 in caso di gestione memoria tramite segmentazione
    - private int numProcessi
      Numero di processi: utile per scegliere il colore per la pagina/segmento

- private int pag_seg
  Boolean che indica se il grafico deve visualizzare (pagine, segmenti)

- private Vector<SquareDraw> pagineSquare = new Vector<SquareDraw>()
  Vector che contiene le pagine/segmenti da visualizzare

- PannelloFrame pannello
  Riferimento alla classe interna incaricata di disegnare le pagine Il JScrollPane si occuperà dello scrolling di questo oggetto, che non è altro che un JPanel.

- Costruttori:
  - public ViewFrameMemoria()
    Unico costruttore incaricato solo di inizializzare il grafico

- **Metodi**:
  - public void aggiorna(Vector<FrameMemoria> memoria, int istante, Vector<Vector<Integer>> processiUltimati) throws Exception
    Funzione che aggiorna il grafico della RAM con i cambiamenti avvenuti nel nuovo istante

  - public void configura(int sceltaGestioneMemoria, int dimMemoria, int numProcessi, int numPagine)
    Metodo che imposta tutte le variabili necessarie per creare il grafico

### 3.3.4.3 PannelloFrame

- Tipo, obiettivo e funzione della classe/interfaccia:
  Eredita da JPanel, incaricata di disegnare al suo interno le pagine/segmenti. E' l'oggetto su cui il JScrollPane fa lo scrolling

- Relazioni d'uso con altre componenti:
  E' la classe interna a ViewFrameMemoria.

- Interfacce e relazioni di uso da altre componenti:
  Nessuna

- Attività svolte e dati trattati:

- Campi dati:
  - Nessuno

- Costruttori:
  - Nessuno

- **Metodi**:
  - private void disegnaPagine(Graphics g)
    Metodo incaricato di disegnare le pagine/segmenti, ognuna con il colore del processo relativo e con il proprio indirizzo logico. Nel caso delle pagine, se queste superano l'area visibile viene aumentata l'altezza del JPanel e comunicata al JScrollPane attraverso il metodo revalidate() per gli opportuni aggiustamenti grafici

  - public void paintComponent(Graphics g)
    Viene usato per disegnare le pagine/segmenti nel JPanel attraverso il metodo disegnaPagine()

#### 3.3.4.4 ViewGrafico

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe visualizza il grafico di tutti i fault di memoria

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del grafico

- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno

  - Costruttori:
    - public ViewGrafico()
      Unico costruttore che crea il pannello del grafico dei fault di memoria

  - **Metodi**:
    - public void aggiornaGrafico(Player player)
      Metodo che aggiorna il grafico

### 3.3.4.5 ViewGraficoTempiAccesso

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe visualizza il grafico di tutti i tempi di accesso

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del grafico

- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno

  - Costruttori:
    - Nessuno

  - **Metodi**:
    - public void aggiorna(XYSeries serie)
      Metodo che aggiorna il grafico

### 3.3.4.6 ViewGraficoTempiBanda

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe visualizza il grafico di tutti i tempi di banda

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del grafico

- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno

  - Costruttori:

- Nessuno

- **Metodi**:
  - public void aggiorna(XYSeries serie)
    Metodo che aggiorna il grafico

### 3.3.4.7 ViewGraficoTempiSlice

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe visualizza il grafico di tutti i tempi slice

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del grafico

- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno

  - Costruttori:
    - Nessuno

  - **Metodi**:
    - public void aggiorna(XYSeries serie)
      Metodo che aggiorna il grafico

### 3.3.4.8 ViewGraficoTempiSwitch

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe visualizza il grafico di tutti i context switch

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del grafico

- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno
  - Costruttori:
    - Nessuno
  - **Metodi**:
    - public void aggiorna(XYSeries serie)
      Metodo che aggiorna il grafico

### 3.3.4.9 ViewGraficoTempiTotali

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe visualizza il grafico di tutti i tempi totali
- Relazioni d'uso con altre componenti:
  Nessuna
- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica del grafico
- Attività svolte e dati trattati:
  - Campi dati:
    - Nessuno
  - Costruttori:
    - Nessuno
  - **Metodi**:
    - public void aggiorna(XYSeries serie)
      Metodo che aggiorna il grafico

### 3.3.4.10 ViewRiepilogo

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe visualizza la configurazione attuale
- **Relazioni d'uso con altre componenti:**

Nessuna

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da SigeMv2View per la creazione della vista statica del riepilogo

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - Nessuno

  - **Costruttori:**
    - public ViewRiepilogo()
      Unico costruttore che crea la vista del riepilogo

  - **Metodi**:
    - public void aggiorna( ConfigurazioneIniziale C )
      Metodo che aggiorna il riepilogo con la configurazione attuale

    - public void azzeraRiepilogo()
      Metodo che inizializza il riepilogo

    - private JTable creaTabella( Processo P )
      Metodo che crea la tabella

    - private void initComponent()
      Metodo che costruisce tutto il frame

### 3.3.4.11 ViewStatistiche

- Tipo, obiettivo e funzione della classe/interfaccia:
  La classe estende JPanel e crea la finestra delle statistiche

- Relazioni d'uso con altre componenti:
  Nessuna

- Interfacce e relazioni di uso da altre componenti:
  Viene usata da SigeMv2View per la creazione della vista statica delle statistiche

- Attività svolte e dati trattati:
  - Campi dati:

- private static final String assunzioni
  Stringa statica

- private ConfigurazioneIniziale configurazioneIniziale
  campo dove contengono tutte le info per la configurazione iniziale

- private LinkedList<Boolean> contextSwitchs
  Lista di booleani che riguardano tutti i context switch

- private LinkedList<Integer> listaTempi
  Lista di tutti i tempi

- private int numeroFault
  Intero dove viene salvato il numero totale di fault

- private Vector<Vector<Integer>> processiUltimati
  Vettore di tutti i processi ultimati

- private int tempo
  Intero dove viene salvato il tempo

- Costruttori:
  - public public ViewStatistiche() ()
    Unico costruttore che crea la vista statica delle statistiche

- **Metodi**:
  - void aggiornaFault(Istante istante, boolean avanti, Player player)
    Aggiorna tutti i fault, istante per istante

  - void aggiornaNumeroIstanti(Player player)
    Aggiorna il numero totale di istanti

  - public void aggiornaTempoTrascorso(Player player,Istante istante, boolean avanti, int numeroIstante)
    Aggiorna il tempo trascorso

  - void aggiornaUtilizzoRAM(Player player,Istante istante)
    Aggiorna l'utilizzo della RAM

- void aggiornaUtilizzoSwap(Player player,Istante istante)
  Metodo che aggiorna l'utilizzo dello swap

- public void azzeraStatistiche(LinkedList<Boolean> contextSwitchs, Vector<Vector<Integer>> processiUltimati,ConfigurazioneIniziale configurazioneIniziale)
  Metodo che inizializza tutte le statistiche

- public void generaStatistiche(Player player,Istante istante,boolean avanti,int numeroIstante)
  Metodo che inizializza le statistiche

- private void initComponents()
  Metodo che costruisce la vista statica

#### 3.3.4.12 ViewStatoAvanzamentoProcessi

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe estende JPanel e crea la finestra del grafico dei processi

- **Relazioni d'uso con altre componenti:**
  Utilizza la classe interna JPanelGrafico

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da SigeMv2View per la creazione della vista statica del grafico dei processi

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - JPanelGrafico x
      Componente JPanelGrafico su cui viene disegnato il grafico

    - JScrollPane base
      Contiene il JPanelGrafico fornendo la funzionalità della scrollbar

    - JLabel jLabel
      Etichetta che contiene una stringa che descrive lo stato della simulazione.

  - **Costruttori:**

- public ViewStatoAvanzamentoProcessi()
  Unico costruttore che crea il pannello del grafico dei processi

- **Metodi**:
  - public void disegnaGrafico(int[] idProcessi)
    Richiama il metodo disegnaJPanelGrafico(int[] idProcessi) della classe interna JPanelGrafico, che disegna il grafico.
    Disegna un insieme di istanti sul grafico.

  - public void initializeViewStatoAvanzamentoProcessi(Vector procNameRef)
    Inizializza il grafico. Questo metodo viene chiamato quando viene creata una simulazione, e imposta il grafico all'istante zero della simulazione.

  - private void resetLabelText()
    Svuota il contenuto della JLabel

  - public void setLabelText(String text)
    Imposta il testo contenuto nella Jlabel

## 3.4 Specifica parte logica

### 3.4.1 logic

#### 3.4.1.1 Processore

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  E' la classe centrale incaricata di creare la simulazione. E' anche il façade che coordina l'attività di Scheduler e GestoreMemoria.

- **Relazioni d'uso con altre componenti:**
  Utilizza i metodi fineSimulazione, getProcessiInArrivo, eseguiAttivazione, getPCBCorrente, eseguiIncremento e getProcessiTerminati della classe Scheduler.
  Utilizza i metodi esegui e notificaProcessoTerminato della classe GestoreMemoria.
  Utilizza il metodo getRifProcesso di PCB e getId di Processo.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalla classe Simulazione che invoca il Processore per essere

creata. In quanto façade non è riferita da altre classi.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private Scheduler scheduler
      Campo dati che specifica lo scheduler da utilizzare per l'ordinamento dei processi.
    - private GestoreMemoria gestoreMemoria
      Campo dati che specifica lo il gestore della memoria
    - private PCB ultimoEseguito
      Riferimento dell'ultimo PCB mandato in esecuzione.
  - **Costruttori:**
    - public Processore(ConfigurazioneIniziale conf)
      Unico costruttore della classe Processore.
      Imposta lo scheduler e il gestore della memoria necessari per la simulazione.
  - **Metodi:**
    - public LinkedList<Istante> creaSimulazione()
      Metodo principale della classe Processore, incaricato di creare tutta la simulazione, rappresentata da una collezione di istanti.
    - private int calcolaFault(LinkedList<Azione> istruzioni)
      Metodo che si occupa di estrarre il numero di fault di pagina da una lista di istruzioni sulla memoria.
    - private boolean controllaSwapPiena(LinkedList<Azione> istruzioni)
      Metodo il cui compito è ritornare tramite un booleano se l'area di Swap è piena o meno.
    - private boolean controllaRAMPiena(LinkedList<Azione> istruzioni)
      Metodo il cui compito è ritornare tramite un booleano se la RAM è piena o meno.
    - private Istante creaIstante(PCB corrente, LinkedList<Azione> istruzioni, boolean nuovoProcesso, boolean SwapPiena)
      Metodo con il compito di creare una istanza della classe Istante

riguardante l'istante corrente della simulazione.

- private LinkedList<FrameMemoria> estraiFrame(PCB corrente)
  Metodo che ha il compito di estrarre i FrameMemoria necessari al processo correntemente in esecuzione nell'istante corrente della simulazione.

### 3.4.2 logic.caricamento

#### 3.4.2.1 GestioneFile

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; si interfaccia con i file per il funzionamento del sistema di simulazione.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dall'interfaccia grafica per salvare e caricare configurazioni da file specificandone il percorso.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private String percosoFileConfigurazione:
      il percorso del file di configurazione.

    - private ConfigurazioneIniziale configurazione:
      la configurazione da salvare/caricare.

  - **Costruttori:**
    - public GestioneFile(String percorsoFileConfigurazione, ConfigurazioneIniziale conf):
      Costruttore della classe. Riceve come parametri il percorso del file in cui verrà salvata/caricata la ConfigurazioneIniziale. Viene inoltre passata la configurazione da salvare.

  - **Metodi:**
    - public String getPercorsoFileConfigurazione():

ritorna il percorso del file in cui verrà salvata/caricata la configurazione.

- public void setPercorsoFileConfigurazione(String nuovoPercorso):
  imposta il percorso del file di configurazione.

- public boolean salvaFileConfigurazione (ConfigurazioneIniziale conf) throws IOException:
  salva la configurazione passata come parametro nel file.

- public ConfigurazioneIniziale caricaFileConfigurazione() throws IOException, ClassNotFoundException:
  carica la configurazione da file e la ritorna al chiamante.

### 3.4.3 logic.gestioneMemoria

#### 3.4.3.1 GestoreMemoria

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe astratta che rappresenta la struttura generica del sistema di gestione della memoria. Riunisce le caratteristiche comuni ed essenziali per tutte le possibili classi ereditate. Non implementa alcun metodo direttamente, poiché le operazioni che dovrà compiere e le strutture dati che dovrà manipolare sono logicamente discrepanti. L'obbiettivo di questa classe è fornire una lista di azioni che rappresenta la sequenza logica delle operazioni che effettuo in memoria.
- **Relazioni d'uso con altre componenti:**
  Poiché non ha campi dati propri ne metodi implementati, essa non è collegata direttamente alcuna componente.

- **Interfacce e relazioni di uso da altre componenti:**
  Questa classe, o le istanze delle sue sottoclassi, sono utilizzate soltanto dalla classe Processore.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - Nessuno

  - **Costruttori:**

- Default Standard

- **Metodi:**
  - public abstract void notificaProcessoTerminato(int id):
    Metodo astratto che notifica, tramite l'identificativo del processo, quando esso è terminato, per liberare la memoria che esso utilizzava.

  - public abstract LinkedList<Azione> esegui( LinkedList<FrameMemoria> ListaFrame, int UT ):
    Metodo astratto che esegue il calcolo delle operazioni in memoria da effettuare affinché il processo pronto possa eseguire.

#### 3.4.3.2 GestoreMemoriaPaginata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe concreta che eredita direttamente dalla classe GestoreMemoria e ne implementa i metodi astratti. Si occupa interamente ed esclusivamente della gestione della memoria paginata, ossia quando l'utente richiede una simulazione di un sistema a memoria virtuale paginata. Questa classe fornisce al Processore una lista di azioni, che descrive precisamente tutte le operazioni che dovranno esser eseguite in memoria nel corso dell'esecuzione della simulazione.
- **Relazioni d'uso con altre componenti:**
  Utilizza tramite l'interfaccia IRimpiazzo, un'istanza della politica di rimpiazzo delle pagine decisa in fase di configurazione del sistema.
  Utilizza inoltre un'istanza della classe RAMPaginata e una della classe SwapPaginata tramite due riferimenti. Legge informazioni dalla classe ConfigurazioneIniziale. Utilizza strutture dati standard dalla libreria “java.util” in particolare LinkedList, Vector e Iterator. Costruisce istanze di AzionePagina.

- **Interfacce e relazioni di uso da altre componenti:**
  Questa classe, una sua istanza per volta, è utilizzata soltanto dalla classe Processore, tramite il metodo esegui.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private IRimpiazzo PoliticaRimpiazzo:
      Mantiene il riferimento alla politica di Rimpiazzo scelta.

- private RAMPaginata MemoriaRam:
  Mentiene il riferimento alla memoria RAM.

- private SwapPaginata MemoriaSwap:
  Mantiene il riferimento alla memoria Swap.

- private int numero_frame_ram:
  Contiene il numero di frame disponibili in RAM.

- private boolean PaginaNulla;
  Segnala quando l'utente sceglie una dimensione di pagina che è pari o inferiore a zero.

- **Costruttori:**
  - public GestoreMemoriaPaginata( ConfigurazioneIniziale C )
    Inizializza tutti i campi dati secondo le scelte derivanti dalla ConfigurazioneIniziale C.

- **Metodi:**
  - public abstract void notificaProcessoTerminato(int id):
    Metodo concreto che notifica, tramite l'identificativo del processo, quando esso è terminato, per liberare le pagine che esso utilizzava.

  - public abstract LinkedList<Azione> esegui( LinkedList<FrameMemoria> ListaFrame, int UT ):
    Metodo pubblico concreto che esegue le operazioni sulle strutture di memoria. In particolare prende come ingressi una lista di pagine le quali dovranno esser caricate in memoria RAM nonché l'unità di tempo di esecuzione. Questo metodo costruirà e ritornerà alla classe Processore la lista delle azioni che ha compiuto per portare la memoria allo stato attuale.
  - private int inserisci( MemoriaPaginata M, FrameMemoria F, int UT ) throws MemoriaEsaurita:
    Metodo privato che inserisce una pagina F nella memoria M, nell'istante UT. Questo metodo potrà sollevare l'eccezione MemoriaEsaurita quando l'inserimento avviene nella memoria di Swap ed essa sarà piena. L'eccezione verrà quindi gestita dal chiamante cioè il metodo esegui. Questo metodo ritorna la posizione di inserimento.
  - private FrameMemoria rimuovi( Memoria M, FrameMemoria F ):
    Metodo privato che rimuove una pagina F dalla memoria M, e ritorna

contemporaneamente la pagina rimossa, che sarà diversa da F nel caso in cui dovrò rimpiazzare una pagina dalla RAM.

#### 3.4.3.3 GestoreMemoriaSegmentata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta. Questa classe eredita direttamente dalla classe GestoreMemoria e quindi ne implementa i metodi astratti. Manipola la memoria secondo la tecnica della segmentazione. Restituisce le operazioni che ad ogni istante della simulazione dovranno essere eseguite nel sistema a segmentazione affinchè ogni processo possa eseguire.

- **Relazioni d'uso con altre componenti:**
  Utilizza tramite l'interfaccia IAllocazione, un'istanza della politica di allocazione dei segmenti decisa in fase di configurazione del sistema.
  Utilizza inoltre un'istanza della classe RAMSegmentata e una della classe SwapSegmentata tramite due riferimenti. Legge informazioni dalla classe ConfigurazioneIniziale. Utilizza strutture dati standard dalla libreria “java.util” in particolare LinkedList, Vector e Iterator. Costruisce istanze di AzioneSegmento.

- **Interfacce e relazioni di uso da altre componenti:**
  E' utilizzata dalla classe Processore tramite il metodo pubblico “esegui”.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private IAllocazione PoliticaAllocazione:
      Mantiene il riferimento alla politica di allocazione scelta.
    - private RAMSegmentata MemoriaRam:
      Riferisce un'istanza di memoria RAM con segmentazione.
    - private SwapSegmentata MemoriaSwap:
      Riferisce un'istanza di memoria Swap con segmentazione.
  - **Costruttori:**
    - public GestoreMemoriaSegmentata( ConfigurazioneIniziale C ):
      Unico costruttore per la classe che prende come parametro il riferimento ad una configurazione iniziale per inizializzare i campi dati e le strutture ad essa collegate.

- **Metodi:**
  - public void notificaProcessoTerminato(int id):
    Metodo pubblico chiamato dal Processore per notificare la terminazione di un processo. Chiama la cancellazione dalla memoria di tutte le risorse che il processo utilizzava tramite il metodo liberaMemoria(id) di Memoria. Si richiede l'identificativo del processo terminato.

  - private FrameMemoria RimuoviFrame( Vector<FrameMemoria> Frames ):
    Metodo privato di utilità interna alla classe che rimuove un segmento dalla RAM quando essa è piena per far posto ad un altro segmento. La rimozione avviene secondo una politica FIFO. Il parametro Frames dovrà contenere la lista di Segmenti presenti in memoria. Il metodo ritornerà un riferimento al Segmento rimosso.
  - private FrameMemoria Inserisci( MemoriaSegmentata M, FrameMemoria F ) throws MemoriaEsaurita:
    Metodo privato di utilità interna alla classe che inserisce il Segmento F nella memoria M. Può rilanciare l'eccezione MemoriaEsaurita nel caso in cui l'aggiunta del segmento in memoria Swap preveda il superamento della quantità di memoria Swap disponibile.
    L'inserimento in RAM avviene secondo la politica di allocazione configurata. Il metodo ritorna il riferimento allo spazio di memoria in cui si è inserito il segmento.
  - private FrameMemoria Rimuovi( MemoriaSegmentata M, FrameMemoria F ):
    Metodo privato di utilità interna alla classe che rimuove il segmento F dalla memoria M nel caso essa sia Swap, in caso contrario richiama il metodo RimuoviFrame. Ritorna il frame effettivamente rimosso.
  - public LinkedList<Azione> esegui( LinkedList<FrameMemoria> ListaSegmenti, int UT ):
    Metodo pubblico di fondamentale importanza. E' il metodo principale per tutto il sistema. E' chiamato dal Processore passandogli la lista di segmenti da caricare in quell'istante. Restituisce una LinkedList di AzioneSegmento che rappresenta la sequenza ordinata di azioni effettuate nella memoria.

#### 3.4.3.4 Azione

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica, astratta. Questa classe rappresenta un'azione generica effettuata in memoria.

- **Relazioni d'uso con altre componenti:**
  Mantiene soltanto un riferimento generico a FrameMemoria coinvolto nell'azione.
- **Interfacce e relazioni di uso da altre componenti**:
  E' utilizzata indirettamente dal GestoreMemoria per il fatto che è classe astratta e quindi non istanziabile.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private int TipoAzione:
      Descrive il tipo di azione secondo la seguente tabella:
      -1 ERRORE FATALE
      0 RAM PIENA
      1 INSERIMENTO IN RAM
      2 INSERIMENTO IN SWAP
      3 RIMOZIONE DA RAM
      4 RIMOZIONE DA SWAP
    - private FrameMemoria Frame:
      Memorizza il riferimento al Frame generico coinvolto nell'azione.
  - **Costruttori:**
    - public Azione( int T, FrameMemoria F ):
      Costruttore pubblico a due parametri. Il primo rappresenta il tipo di azione, il secondo il frame coinvolto.
    -
  - **Metodi:**
    - public int getAzione():
      Ritorna il contenuto del campo dati TipoAzione.
    - public FrameMemoria getFrame():
      Ritorna il riferimento al FrameMemoria.

#### 3.4.3.5 AzionePagina

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica, concreta. Questa classe eredita da Azione generica effettuata in memoria e ne specializza l'uso per una memoria paginata.
- **Relazioni d'uso con altre componenti:**
  Nessuna oltre a quelle della classe padre.
- **Interfacce e relazioni di uso da altre componenti**:
  E' utilizzata direttamente ed esclusivamente dal GestoreMemoriaPaginata.
- **Attività svolte e dati trattati:**

- Campi dati:
  - private int Pos:
    Mantiene la posizione in memoria dove la pagina è stata inserita.
- **Costruttori:**
  - public Azione( int T, FrameMemoria F, int P):
    Costruttore pubblico a due parametri. Il primo rappresenta il tipo di azione, il secondo il frame coinvolto e il terzo la posizione. Richiama il costruttore della classe padre.

  - public Azione( int T, FrameMemoria F )
    Costruttore pubblico a due parametri. Richiama il costruttore della classe padre. Il campo Pos assume il valore -1 per default.

- **Metodi:**
  - public int getPosizione():
    Ritorna il contenuto del campo dati Pos.

#### 3.4.3.6 AzioneSegmento

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica, concreta. Questa classe eredita da Azione generica effettuata in memoria e ne specializza l'uso per una memoria segmentata.
- **Relazioni d'uso con altre componenti:**
  Nessuna oltre a quelle della classe padre.
- **Interfacce e relazioni di uso da altre componenti**:
  E' utilizzata direttamente ed esclusivamente dal GestoreMemoriaSegmentata.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private FrameMemoria Destinazione:
      Mantiene il riferimento al frame di destinazione in memoria dove il segmento è stato inserito.
  - **Costruttori:**
    - public AzioneSegmento( int T, FrameMemoria F, FrameMemoria D ):
      Costruttore pubblico a tre parametri. Il primo rappresenta il tipo di azione, il secondo il frame coinvolto e il terzo il frame destinazione. Richiama il costruttore della classe padre.
    - public AzionePagina( int T, FrameMemoria F):
      Costruttore pubblico a due parametri. Richiama il costruttore della classe padre. Il campo Destinazione assume il valore “null” per

default.

- **Metodi:**
  - public int getDestinazione():
    Ritorna il contenuto del campo dati Destinazione.

### 3.4.3.7 IAllocazione

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Interfaccia pubblica che definisce la struttura comune a tutte le politiche di allocazione dei segmenti.
- **Relazioni d'uso con altre componenti:**
  Nessuna.
- **Interfacce e relazioni di uso da altre componenti**:
  E' utilizzata esclusivamente dal GestoreMemoriaSegmentata.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - Nessuno.
  - **Costruttori:**
    - Nessuno.
  - **Metodi:**
    - public FrameMemoria Alloca ( FrameMemoria F, Vector<FrameMemoria> Liberi ):
      Sarà implementato da ogni politica di allocazione. Il metodo dovrà restituire il FrameMemoria libero che secondo la politica rispecchia meglio le caratteristiche del frame passatogli F. Liberi è un vettore il quale contiene tutti i FrameMemoria liberi all'interno della memoria RAM.

### 3.4.3.8 IRimpiazzo

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Interfaccia pubblica che definisce la struttura comune a tutte le politiche di rimpiazzo delle pagine.
- **Relazioni d'uso con altre componenti:**
  Nessuna.
- **Interfacce e relazioni di uso da altre componenti**:
  E' utilizzata esclusivamente dal GestoreMemoriaPaginata.
- **Attività svolte e dati trattati:**
  - Campi dati:

- Nessuno.
- **Costruttori:**
  - Nessuno.
- **Metodi:**
  - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
    Implementato dalle politiche di rimpiazzo servirà per aggiornare i dati derivati dall'inserimento in RAM di una pagina.
  - public void LiberaEntry( int Posizione ):
    Resetta i dati in fase di rimozione di una pagina.
  - public FrameMemoria SelezionaEntry():
    Seleziona la pagina da rimuovere secondo la politica prescelta.
  - public void AggiornaEntry( int Posizione, boolean M ):
    Aggiorna i dati dopo un accesso alla pagina da parte di un processo.
  - public void AggiornaEntries( ):
    Aggiorna i dati di tutte le pagine in memoria quando il gestore della memoria lo richiede ai fini del corretto funzionamento degli algoritmi di rimpiazzo.

#### 3.4.3.9 FirstFit,BestFit,NextFit,QuickFit,WorstFit

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classi pubbliche concrete che implementano l'interfaccia IAllocazione. In particolare implementano il metodo Alloca secondo la politica dalla quale prende il nome la classe, rispettivamente:
  FirstFit: Restituisce un riferimento al primo spazio di memoria disponibile a contenere il segmento.
  BestFit: Restituisce un riferimento allo spazio di memoria le cui dimensioni sono il più possibile vicine al segmento da inserire.
  NextFit: Come FirstFit soltanto inizia la ricerca dall'ultimo elemento visitato.
  QuickFit: Come BestFit salvo diversa implementazione.
  WorstFit: Restituisce un riferimento allo spazio di dimensione maggiore.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util.
- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaSegmentata in fase di allocazione dei segmenti. .
- **Attività svolte e dati trattati:**
  - Campi dati:
    - Nessuno, eccetto NextFit che mantiene un riferimento (FrameMemoria Next)

allo spazio libero successivo dal quale iniziare la ricerca.

- **Costruttori:**
  - Nessuno.
- **Metodi:**
  - public FrameMemoria Alloca ( FrameMemoria F, Vector<FrameMemoria> Liberi ):
    Implementa l'algoritmo di allocazione secondo quanto decritto nell'obbiettivo della classe, restituendo un riferimento allo spazio sul quale effettuare l'allocazione. Come parametri richiede un riferimento F al segmento da inserire e un Vector di riferimenti agli spazi liberi in memoria.

### 3.4.3.10 A

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di A per Anging cioè implementa l'algoritmo dell'invecchiamento.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.
- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private class Dati:
      Classe interna privata che memorizza un intero Contatore che memorizza l'età di una pagina, un boolean R che indica se recentemente la pagina è stata riferita e il riferimento F alla pagina in questione.
    - private Vector<Dati> Tabella:
      Vettore privato che memorizza una sequenza di Dati come immagine della memoria.
  - **Costruttori:**
    - public A( int dim ):
      Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default.
  - **Metodi:**
    - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):

Viene richiamato quando carico in memoria RAM una pagina. Inizializza i dati relativi a quella pagina, in particolare F è il riferiferimento alla pagina mentre gli altri parametri sono inutilizzati.

- public void LiberaEntry( int Posizione ):
  Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
- public FrameMemoria SelezionaEntry():
  Implementa l'algoritmo vero e proprio. Semplicemente restituisce un riferimento alla pagina il cui Contatore è il minore possibile, ossia la pagina memo utilizzata.
- public void AggiornaEntry( int Posizione, boolean M ):
  Imposta R a true quando una pagina la pagina nella posizione specificata è stata riferita. Il Parametro M è inutilizzato in questo algoritmo.
- public void AggiornaEntries():
  Per ogni pagina, imposta il campo R a false e incrementa il contatore se la pagina era recentemente riferita altrimenti lo decrementa.

#### 3.4.3.11 C

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di C per Clock cioè implementa l'algoritmo dell'orologio.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.
- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private class Dati:
      Classe interna privata che memorizza un boolean R che indica se recentemente la pagina è stata riferita e il riferimento F alla pagina in questione.
    - private Vector<Dati> Tabella:
      Vettore privato che memorizza una sequenza di Dati come immagine della memoria.
    - private int Lancetta:
      Memorizza il punto in cui era arrivata la lancetta dell'orologio.
  - **Costruttori:**

- public C( int dim ):
  Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default.

- **Metodi:**
  - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
    Viene richiamato quando carico in memoria RAM una pagina. Inizializza i dati relativi a quella pagina, in particolare F è il riferiferimento alla pagina e imposta R a true, mentre gli altri parametri sono inutilizzati.
  - public void LiberaEntry( int Posizione ):
    Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
  - public FrameMemoria SelezionaEntry():
    Implementa l'algoritmo vero e proprio. Semplicemente restituisce un riferimento alla pagina indicata dalla lancetta se essa non è stata recentemente riferita altrimenti imposta R a false e avanza la lancetta di una posizione per ricompiere il controllo.
  - public void AggiornaEntry( int Posizione, boolean M ):
    Non fa nulla in questo algoritmo.
  - public void AggiornaEntries():
    Non fa nulla in questo algoritmo.

#### 3.4.3.12 FIFO

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di FIFO per FirstInFirstOut cioè rimpiazza la pagina che è entrata per prima in memoria sebbene questa sia stata utilizzata di recente.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.
- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private class Dati:
      Classe interna privata che memorizza in un intero TArrivo l'unità di tempo in cui la pagina è stata caricata in RAM, oltre che il riferimento F alla pagina.

Si implementa anche un metodo private boolean Minore( Dati D ) che confronta due Dati e ne indica se uno è minore dell'altro.

- private Vector<Dati> Tabella:
  Vettore privato che memorizza una sequenza di Dati come immagine della memoria.

- **Costruttori:**
  - public FIFO( int dim ):
    Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default.
- **Metodi:**
  - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
    Imposta F riferimento alla pagina e imposta Tarrivo a UT, mentre gli altri parametri sono inutilizzati.
  - public void LiberaEntry( int Posizione ):
    Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
  - public FrameMemoria SelezionaEntry():
    Implementa l'algoritmo. Restituisce un riferimento alla pagina il cui Tarrivo è minore.
  - public void AggiornaEntry( int Posizione, boolean M ):
    Non fa nulla in questo algoritmo.
  - public void AggiornaEntries():
    Non fa nulla in questo algoritmo.

#### 3.4.3.13 LRU

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di LRU per Least Recently Used cioè rimpiazza la pagina usata meno di recente.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.
- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private int Contatore:

Memorizza un contatore incrementato ad ogni riferimento in memoria.

- private class Dati:
  Classe interna privata che memorizza in un intero UltimoAccesso il Contatore corrente, oltre che il riferimento F alla pagina.
- private Vector<Dati> Tabella:
  Vettore privato che memorizza una sequenza di Dati come immagine della memoria.

- **Costruttori:**
  - public LRU( int dim ):
    Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default e inizializza il Contatore a 0.
- **Metodi:**
  - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
    Imposta F riferiferimento alla pagina nella Posizione indicata, e imposta UltimoAccesso con il valore corrente del Contatore, incrementandolo successivamente.
  - public void LiberaEntry( int Posizione ):
    Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
  - public FrameMemoria SelezionaEntry():
    Implementa l'algoritmo. Restituisce un riferimento alla pagina il cui valore di UltimoAccesso è minore.
  - public void AggiornaEntry( int Posizione, boolean M ):
    Aggiorna l'UltimoAccesso della pagina in Posizione e incrementa Contatore.
  - public void AggiornaEntries():
    Non fa nulla in questo algoritmo.

### 3.4.3.14 NFU

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di NFU per Not Frequently Used cioè rimpiazza la pagina usata meno di frequente.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.
- **Interfacce e relazioni di uso da altre componenti**:

Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.

- **Attività svolte e dati trattati:**
  - Campi dati:
    - private class Dati:
      Classe interna privata che memorizza in un intero Contatore il numero di utilizzi della pagina, in un boolean R se è stata riferita di recente, infine il riferimento F alla pagina.
    - private Vector<Dati> Tabella:
      Vettore privato che memorizza una sequenza di Dati come immagine della memoria.
  - **Costruttori:**
    - public NFU( int dim ):
      Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default.
  - **Metodi:**
    - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
      Imposta F riferimento alla pagina nella Posizione indicata, imposta il Contatore a zero e R a true.
    - public void LiberaEntry( int Posizione ):
      Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
    - public FrameMemoria SelezionaEntry():
      Implementa l'algoritmo. Restituisce un riferimento alla pagina il cui valore di Contatore è minore.
    - public void AggiornaEntry( int Posizione, boolean M ):
      Imposta il campo R a true significando che la pagina è stata riferita.
    - public void AggiornaEntries():
      Per ogni pagina se essa è stata riferita, cioè R è true, incremento il Contatore e resetto R.

### 3.4.3.15 NRU

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di NRU per Not Recently Used cioè rimpiazza la pagina non usata di recente.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una

struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.

- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private class Dati:
      Classe interna privata che memorizza un boolean M che indica se la pagina è stata modificata, un boolean R che indica se la pagina è stata riferita, oltre al riferimento F alla pagina. Si implementa un metodo int Classe() che ritorna la classe a cui appartiene una pagina:
      R=false M=false Classe=0
      R=false M=true Classe=1
      R=true M=false Classe=2
      R=true M=true Classe=3
    - private Vector<Dati> Tabella:
      Vettore privato che memorizza una sequenza di Dati come immagine della memoria.
  - **Costruttori:**
    - public NRU( int dim ):
      Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default.
  - **Metodi:**
    - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
      Imposta F riferimento alla pagina nella Posizione indicata, imposta R a true e M con il valore del parametro passato.
    - public void LiberaEntry( int Posizione ):
      Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
    - public FrameMemoria SelezionaEntry():
      Implementa l'algoritmo. Restituisce un riferimento alla pagina di Classe minore.
    - public void AggiornaEntry( int Posizione, boolean M ):
      Aggiorna R e M della pagina in Posizione.
    - public void AggiornaEntries():
      Per ogni pagina resetta R a false.

### 3.4.3.16 SC

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica concreta che implementa l'interfaccia IRimpiazzo. Essa prende il nome di SC per Second Chance cioè FIFO soltanto con controllo di riferimento.
- **Relazioni d'uso con altre componenti:**
  Utilizzano la classe Vector standard dalla libreria java.util per mantenere una struttura di Dati con i quali effettuare decisioni per il rimpiazzo delle pagine.
- **Interfacce e relazioni di uso da altre componenti**:
  Sono utilizzate esclusivamente dal GestoreMemoriaPaginata in fase di rimpiazzo delle pagine.
- **Attività svolte e dati trattati:**
  - Campi dati:
    - private class Dati:
      Classe interna privata che memorizza in un intero TArrivo l'unità di tempo in cui la pagina è stata caricata in RAM, un boolean R che indica se la pagina è stata recentemente riferita, oltre al riferimento F alla pagina.
    - private Vector<Dati> Tabella:
      Vettore privato che memorizza una sequenza di Dati come immagine della memoria.
  - **Costruttori:**
    - public SC( int dim ):
      Data la dimensione intera della memoria interpretata come il numero di FrameMemoria, istanzia Tabella con valori di default e inizializza il Contatore a 0.
  - **Metodi:**
    - public void InserisciEntry( FrameMemoria F, int Posizione, int UT, boolean M ):
      Imposta F riferimento alla pagina nella Posizione indicata, e imposta TArrivo con il valore corrente di UT, R ovviamente sarà true.
    - public void LiberaEntry( int Posizione ):
      Resetta i campi dati relativi alla pagina nella Posizione specificata dal parametro.
    - public FrameMemoria SelezionaEntry():
      Implementa l'algoritmo. Restituisce un riferimento alla pagina il cui valore di TArrivo è minore e contemporaneamente il campo R sia false.
    - public void AggiornaEntry( int Posizione, boolean M ):

Non esegue nulla in questo algoritmo.

- public void AggiornaEntries():
  Non esegue nulla in questo algoritmo.

### 3.4.3.17 RAMPaginata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe che nel nostro sistema modella la RAM sotto forma di collezione di pagine. E' quindi l'implementazione della RAM che verrà usata quando l'utente sceglie come tecnica di gestione della memoria la paginazione. La classe offre tutti i metodi necessari per l'inserimento, la rimozione e la ricerca delle pagine.

- **Relazioni d'uso con altre componenti:**
  Eredita direttamente da MemoriaPaginata, classe base per il tipo di memorie modellate come collezioni di pagine. Ovviamente usa la classe Pagina e i suoi metodi, anche per gestire le pagine in memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  La classe viene usata esclusivamente dalla classe GestoreMemoriaPaginata.

- **Attività svolte e dati trattati:**
  - **Costruttori:**
    - public RAMPaginata(ConfigurazioneIniziale conf) throws PaginaNulla:

      Unico costruttore, attraverso il riferimento all'istanza di ConfigurazioneIniziale imposta la dimensione della RAM quantificata come numero di pagine. In caso di errata dimensione della pagina (<=0) lancia l'eccezione PaginaNulla.

  - **Metodi**:
    - public int aggiungi(FrameMemoria pag) throws MemoriaEsaurita:
      Metodo che aggiunge una pagina nella RAM. Se ha successo, ritorna un int che rappresenta l'id della "cella" in cui ha inserito la pagina, quindi l'indice del Vector memoria in cui il riferimento è stato inserito; in caso di fallimento solleva un'eccezione che deve essere opportunamente gestita. Richiede un parametro di tipo FrameMemoria, il riferimento alla pagina da aggiungere in RAM.

    - public boolean cerca(FrameMemoria pag):

Metodo che cerca se una pagina è già presente in RAM: serve per verificare se scatta un page fault o meno. Chiede come parametro il riferimento alla pagina.

- public int indiceDi(FrameMemoria pag):
  Metodo che, dato un riferimento di tipo FrameMemoria, ritorna un int rappresentante la sua posizione all'interno della RAM. Utile in qualche politica di rimpiazzo delle pagine.

- public void liberaMemoria(int idProcesso):
  Metodo che marca come eliminabili le pagine riferite da un processo che finito la sua esecuzione. Durante il prossimo inserimento di una pagina, queste pagine saranno considerate memoria libera, quindi si potranno inserire nuove pagine senza interrogare la politica di rimpiazzo. Richiede come parametro un int, l'identificativo univoco del processo che ha finito la sua esecuzione.

- public boolean rimuovi(FrameMemoria pag):
  Metodo che marca la pagina come non più in RAM, liberando spazio. Questo metodo viene usato subito dopo aver scelto la pagina da rimpiazzare: quest'ultima è quella riferita dal parametro pag.

#### 3.4.3.18 SwapPaginata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe che nel nostro sistema modella lo swap sotto forma di collezione di pagine. E' quindi l'implementazione dello swap che verrà usata quando l'utente sceglie come tecnica di gestione della memoria la paginazione. La classe offre tutti i metodi necessari per l'inserimento, la rimozione e la ricerca delle pagine.

- **Relazioni d'uso con altre componenti:**
  Eredita direttamente da MemoriaPaginata, classe base per il tipo di memorie modellate come collezioni di pagine. Ovviamente usa la classe Pagina e i suoi metodi, anche per gestire le pagine in memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  La classe viene usata esclusivamente dalla classe GestoreMemoriaPaginata.

- **Attività svolte e dati trattati:**

- **Costruttori:**
  - public SwapPaginata(ConfigurazioneIniziale conf) throws PaginaNulla:
    Unico costruttore, attraverso il riferimento all'istanza di ConfigurazioneIniziale imposta la dimensione dello swap quantificata come numero di pagine. Lancia l'eccezione PaginaNulla nel caso di errata impostazione della grandezza della pagina (<=0).

- **Metodi:**
  - public int aggiungi(FrameMemoria pag) throws MemoriaEsaurita:
    Metodo che aggiunge una pagina nello swap. Se la memoria non è piena, aggiunge la pagina; altrimenti lancia un'eccezione e a quel punto il programma termina (se lo swap è pieno, lo è di conseguenza anche la RAM e ciò provoca il termine del programma). Come parametro riceve il riferimento alla pagina da inserire.

  - public void liberaMemoria(int idProcesso):
    Metodo che rimuove dallo swap le pagine riferite da un processo che ha finito la sua esecuzione. Richiede come parametro un int, l'identificativo univoco del processo che ha finito la sua esecuzione.

  - public boolean rimuovi(FrameMemoria pag):
    Metodo che rimuove dallo swap la pagina riferita dal parametro pag.

### 3.4.3.19 RAMSegmentata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe che nel nostro sistema modella la RAM sotto forma di collezione di segmenti. E' quindi l'implementazione della RAM che verrà usata quando l'utente sceglie come tecnica di gestione della memoria la segmentazione. La classe offre tutti i metodi necessari per l'inserimento, la rimozione e la ricerca dei segmenti.

- **Relazioni d'uso con altre componenti:**
  Eredita direttamente da MemoriaSegmentata, classe base per il tipo di memorie modellate come collezioni di segmenti. Ovviamente usa la classe Segmento e i suoi metodi, anche per gestire i segmenti in memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  La classe viene usata esclusivamente dalla classe GestoreMemoriaSegmentata.

- **Attività svolte e dati trattati:**
  - **Costruttori:**
    - public RAMSegmentata(ConfigurazioneIniziale conf):
      Unico costruttore, attraverso il riferimento all'istanza di ConfigurazioneIniziale imposta la dimensione della RAM in termini di byte.

  - **Metodi:**
    - public void aggiungi(FrameMemoria seg, FrameMemoria spazio):
      Metodo che aggiunge un segmento nella RAM. Prende come parametri il riferimento al segmento da inserire e il riferimento allo spazio scelto dalla politica di allocazione dei segmenti.

    - public boolean cerca(FrameMemoria seg):
      Metodo che cerca se un segmento è già presente in RAM: serve per verificare se scatta un page fault o meno. Chiede come parametro il riferimento al segmento.

    - public void liberaMemoria(int idProcesso):
      Metodo che elimina i segmenti non più riferiti perchè il relativo processo ha terminato la sua esecuzione. Nel caso i segmenti da eliminare siano contigui a spazi vuoti, viene creato un unico spazio di dimensione uguale alla somma delle grandezze di spazi e segmento. Prende come parametro l'identificativo univoco del processo che ha finito di eseguire.

    - public boolean rimuovi(FrameMemoria seg):
      Metodo che rimuove il segmento riferito da seg. Nel caso il segmento da eliminare sia contiguo a spazi vuoti, viene creato un unico spazio di dimensione uguale alla somma delle grandezze di spazi e segmento.

    - public Vector<FrameMemoria> getFrameLiberi():
      Metodo che costruisce e restituisce un Vector composto dagli spazi vuoti disponibili in RAM. Utile per gli algoritmi di allocazione dei segmenti.

    - public Vector<FrameMemoria> getFrameOccupati():
      Metodo che costruisce e restituisce un Vector composto dai segmenti presenti in RAM. Utile per gli algoritmi di rimpiazzo dei segmenti.

- public FrameMemoria getSpazioMaggiore():
  Metodo che restituisce il segmento con lo spazio libero più grande. Utile per gli algoritmi di rimpiazzo dei segmenti.
- public int indiceDi(FrameMemoria seg)
  Metodo che, dato un frame memoria, ritorna la sua posizione all'interno della RAM.

### 3.4.3.20 SwapSegmentata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe che nel nostro sistema modella lo swap sotto forma di collezione di segmenti. E' quindi l'implementazione dello swap che verrà usata quando l'utente sceglie come tecnica di gestione della memoria la segmentazione. La classe offre tutti i metodi necessari per l'inserimento, la rimozione e la ricerca dei segmenti.

- **Relazioni d'uso con altre componenti:**
  Eredita direttamente da MemoriaSegmentata, classe base per il tipo di memorie modellate come collezioni di segmenti. Ovviamente usa la classe Segmento e i suoi metodi, anche per gestire i segmenti in memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  La classe viene usata esclusivamente dalla classe GestoreMemoriaSegmentata.

- **Attività svolte e dati trattati:**
  - **Costruttori:**
    - public SwapSegmentata(ConfigurazioneIniziale conf):
      Unico costruttore, attraverso il riferimento all'istanza di ConfigurazioneIniziale imposta la dimensione dello swap in termini di byte.

  - **Metodi:**
    - public void aggiungi(FrameMemoria seg) throws MemoriaEsaurita:
      Metodo che aggiunge un segmento nello swap. Se la memoria non è piena, aggiunge il segmento; altrimenti lancia un'eccezione e a quel punto il programma termina (se lo swap è pieno, lo è di conseguenza anche la RAM e ciò provoca il termine del programma). Come parametro riceve il riferimento al segmento da inserire.

- public void liberaMemoria(int idProcesso):
  Metodo che rimuove dallo swap i segmenti riferiti da un processo che ha finito la sua esecuzione. Richiede come parametro un int, l'identificativo univoco del processo che ha finito la sua esecuzione.

- public boolean rimuovi(FrameMemoria seg):
  Metodo che rimuove dallo swap la pagina riferita dal parametro seg.

#### 3.4.3.21 Memoria

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe astratta che rappresenta una generica memoria: verrà poi implementata da tutte le memorie, sia paginate che segmentate.

- **Relazioni d'uso con altre componenti:**
  La classe contiene al suo interno la struttura dati essenziale per realizzare le memorie, costituita dall'utilizzo della classe java.util.Vector istanziata con FrameMemoria.

- **Interfacce e relazioni di uso da altre componenti:**
  Questa classe è la classe base per tutte le memorie, perciò viene ereditata direttamente sia da MemoriaPaginata sia da MemoriaSegmentata. Indirettamente, viene utilizzata da tutte le successive istanziazioni delle memorie.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected Vector<FrameMemoria> memoria:
      Rappresenta la struttura dati che è il cuore di ogni memoria. E' incaricata di mantenere i riferimenti ai FrameMemoria.

  - **Metodi**:
    - public abstract boolean rimuovi(FrameMemoria frame):
      Metodo astratto la cui implementazione dovrà rimuovere un FrameMemoria dalla memoria. Il FrameMemoria da togliere è quello riferito dal parametro richiesto frame. Ritorna TRUE se la rimozione ha successo, FALSE altrimenti.

    - public abstract boolean cerca(FrameMemoria frame):

Metodo astratto la cui implementazione dovrà cercare un FrameMemoria nella memoria. frame è il riferimento al FrameMemoria da cercare. Ritorna TRUE se il frame è in memoria, FALSE altrimenti.

- public abstract void liberaMemoria(int idProcesso):
  Metodo astratto la cui implementazione dovrà rimuovere tutti i FrameMemoria riferiti da un processo che ha finito la sua esecuzione. Richiede come parametro un int che è l'identificativo univoco del processo che ha finito di eseguire.

### 3.4.3.22 MemoriaPaginata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe astratta che rappresenta le memorie che utilizzano paginazione. Verrà concretizzata poi dalle classi RAMPaginata, SwapPaginata.

- **Relazioni d'uso con altre componenti:**
  Eredita direttamente da Memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  Questa classe astratta verrà implementata e usata concretamente esclusivamente da RAMPaginata e SwapPaginata.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected int pagineResidue:
      Campo dati che indica il numero di pagine che la memoria può ancora contenere.
  - **Costruttori:**
    - public MemoriaPaginata(int dimMemoria, int dimPagina) throws PaginaNulla:
      Costruttore di MemoriaPaginata: si limita a inizializzare il numero di pagine residue in memoria. Lancia un'eccezione nel caso in cui la dimensione della pagina sia stata impostata a 0. Richiede due parametri: dimMemoria che rappresenta la dimensione della memoria espressa in byte, dimPagina che rappresenta la dimensione della pagina espressa in byte.
  - **Metodi:**
    - public abstract int aggiungi(FrameMemoria frame) throws MemoriaEsaurita:

Metodo astratto la cui implementazione nelle sottoclassi dovrà permettere l'inserimento di una pagina in memoria. Richiede come parametro un riferimento alla pagina da inserire. Ritorna un int rappresentante la posizione di inserimento del frame all'interno del Vector memoria. In caso di spazio insufficiente, è previsto il lancio di un'eccezione che dovrà essere opportunamente gestita.

#### 3.4.3.23 MemoriaSegmentata

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe astratta che rappresenta le memorie che utilizzano segmentazione. Verrà concretizzata poi dalle classi RAMSegmentata, SwapSegmentata.

- **Relazioni d'uso con altre componenti:**
  Eredita direttamente da Memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  Questa classe astratta verrà implementata e usata concretamente esclusivamente da RAMSegmentata e SwapSegmentata.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected int spazioResiduo:
      Campo dati di tipo int che indica lo spazio residuo della RAM espresso in byte.

  - **Costruttori:**
    - public MemoriaSegmentata(int spazio):
      Costruttore di MemoriaPaginata: si limita a inizializzare lo spazio residuo della memoria espresso in byte.

    - public abstract void aggiungi(FrameMemoria frame, FrameMemoria spazio) throws MemoriaEsaurita:
      Metodo astratto la cui implementazione nelle sottoclassi dovrà permettere l'aggiunta di un segmento in memoria. Chiede come parametro il riferimento al segmento da inserire. Può lanciare l'eccezione MemoriaEsaurita nel caso in cui non ci sia più spazio per inserire il segmento.

### 3.4.3.24 PaginaNulla

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe rappresenta un'eccezione lanciata quando si cerca di inizializzare una memoria avendo impostato la dimensione della pagina a 0.

- **Interfacce e relazioni di uso da altre componenti:**
  L'eccezione è lanciata esclusivamente dalle memorie che usano paginazione.

### 3.4.3.25 MemoriaEsaurita

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe rappresenta un'eccezione lanciata quando si cerca di inserire un FrameMemoria in una memoria già piena. Restituisce anche la dimensione residua della memoria (utile soltanto in caso di segmentazione).

- **Interfacce e relazioni di uso da altre componenti:**
  L'eccezione è lanciata da tutte le classi che rappresentano le memorie e implementano quindi la classe Memoria.

- **Attività svolte e dati trattati:**
  - **Campi dato:**
    - private int spazioResiduo:
      Intero che rappresenta lo spazio residuo espresso in byte rimanente in Memoria.

  - **Costruttori:**
    - public MemoriaEsaurita(int dimensione):
      Costruttore dell'eccezione MemoriaEsaurita. Chiede come parametro un intero che rappresenta lo spazio residuo rimasto in memoria.

  - **Metodi:**
    - public int getSpazioResiduo():
      Funzione che ritorna un intero rappresentante lo spazio residuo espresso in byte.

### 3.4.3.26 FrameMemoria

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Interfaccia pubblica; racchiude gli aspetti comuni di pagine e segmenti.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene implementata dalle classi: Pagina e Segmento.
  Viene utilizzata in tutte le classi che compongono e interagiscono con il gestore della memoria.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    Nessuno

  - **Costruttori:**
    Nessuno.

  - **Metodi:**
    - public String getIndirizzo():
      ritorna l'indirizzo del FrameMemoria.

    - public int getDimensione():
      ritorna la dimensione del FrameMemoria.

    - public boolean getInRAM():
      ritorna un booleano che indica se il FrameMemoria è in RAM.

    - public boolean setInRAM(boolean nuovoStato):
      specifica se il FrameMemoria è in RAM.

    - public boolean getSolaLettura():
      ritorna un booleano che indica se il FrameMemoria è di sola lettura.

    - public boolean setSolaLettura(boolean nuovoStato):
      marca/smarca il FrameMemoria in sola lettura.

    - public int getIdProcesso():
      ritorna l'id del processo che detiene il FrameMemoria.

    - public boolean setIdProcesso(int idProcessoPassato):
      imposta il processo che detiene il FrameMemoria.

- public boolean getModifica():
  ritorna un booleano che indica se il FrameMemoria è stato modificato.

- public boolean setModifica(boolean nuovoStato):
  marca/smarca il FrameMemoria come modificato.

#### 3.4.3.27 Pagina

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; rappresenta una pagina che può risiedere in memoria. Implementa l'interfaccia FrameMemoria.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dal gestore della memoria e da tutte le classi che interagiscono con lui.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private boolean solaLettura:
      indica se la pagina è di sola lettura.

    - private boolean bloccata:
      indica se la pagina è bloccata.

    - private boolean modificata:
      indica se la pagina è modificata.

    - private final String INDIRIZZO:
      l'indirizzo della pagina.

    - private static int DIMENSIONE:
      la dimensione della pagina.

    - private boolean inRAM:
      indica se la pagina è in RAM.

    - private int idProcesso:

l'id del processo che possiede la pagina.

- **Costruttori:**
  - public Pagina(String indirizzoPagina,int dimensione,int idProcesso):
    Costruttore della classe. Crea una pagina specificando il suo indirizzo, la dimensione e l'id del processo che la possiede.

- **Metodi:**
  - public boolean getSolaLettura():
    ritorna un booleano che indica se una pagina è di sola lettura.
  - public boolean setSolaLettura(boolean nuovoStato):
    marca/smarca una pagina in stato di sola lettura.
  - public boolean getBloccata():
    ritorna un booleano che indica se una pagina è bloccata.
  - public boolean setBloccata(boolean nuovoStato):
    marca/smarca una pagina come bloccata.
  - public String getIndirizzo():
    ritorna l'indirizzo della pagina in memoria.
  - public boolean getInRAM():
    ritorna un booleano che indica se una pagina è in RAM.
  - public boolean setInRAM(boolean nuovoStato):
    marca/smarca una pagina come in RAM.
  - public int getDimensione():
    ritorna la dimensione di una pagina.
  - public int getIdProcesso():
    ritorna l'id del processo che detiene la pagina.
  - public boolean setIdProcesso(int idProcessoPassato):
    imposta il processo che detiene la pagina.
  - public boolean getModifica():

ritorna un booleano che indica se una pagina è modificata.

- public boolean setModifica(boolean nuovoStato):
  marca/smarca una pagina come modificata.

#### 3.4.3.28 Segmento

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; rappresenta un segmento che può risiedere in memoria. Implementa l'interfaccia FrameMemoria.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dal gestore della memoria e da tutte le classi che interagiscono con lui.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private boolean solaLettura:
      indica se il segmento è di sola lettura.
    - private boolean modificato:
      indica se il segmento è modificato.
    - private final String INDIRIZZO:
      l'indirizzo del segmento.
    - private int dimensione:
      la dimensione del segmento.
    - private boolean inRAM:
      indica se il segmento è in RAM.
    - private int idProcesso:
      l'id del processo che possiede il segmento.
    - private int tempoInRAM:
      il tempo trascorso da quando il segmento è in RAM.

- **Costruttori:**
  - public Segmento(String indirizzo, int dimensione, int idProcesso):
    Costruttore della classe. Crea un segmento specificando l'indirizzo, la dimensione e l'id del processo che lo possiede.

- **Metodi:**
  - public boolean getSolaLettura()
    ritorna un booleano che indica se il segmento è di sola lettura.
  - public boolean setSolaLettura(boolean nuovoStato)
    marca/smarca un segmento come sola lettura.
  - public int getDimensione()
    ritorna la dimensione del segmento.
  - public String getIndirizzo()
    ritorna l'indirizzo del segmento.
  - public boolean getInRAM()
    ritorna un booleano che indica se il segmento è in RAM.
  - public boolean setInRAM(boolean nuovoStato)
    marca/smarca un segmento come in RAM.
  - public int getIdProcesso()
    ritorna l'id del processo che possiede il segmento.
  - public boolean setIdProcesso(int idProcessoPassato)
    imposta il processo che detiene il segmento.
  - public boolean getModifica()
    ritorna un booleano che indica se il segmento è modificato.
  - public boolean setModifica(boolean nuovoStato)
    marca/smarca un segmento come modificato.
  - public int getTempoInRam()
    ritorna il tempo trascorso da quando il segmento è in RAM.

- public boolean setTempoInRAM(int nuovoTempo)
  imposta il tempo trascorso da quando il segmento è in RAM.

### 3.4.4 logic.parametri

#### 3.4.4.1 ConfigurazioneIniziale

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta. Rappresenta la configurazione del sistema di simulazione. Estende la classe Serializable per permettere la serializzazione in file.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalle classi: SwapPaginata, GestoreMemoriaPaginata, Simulazione, Processore, RAMSegmentata, Player, GestioneFile, AssociazioneProcessiJDialog, RAMPaginata, GestoreMemoriaSegmentata, SwapSegmentata.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private final int BANDA_BUS_DATI:
      capacità del bus-dati espressa in KB.

    - private final int POLITICA_GESTIONE_MEMORIA:
      specifica la politica di gestione della memoria (vedi Analisi dei Requisiti).

    - private final int MODALITA_GESTIONE_MEMORIA:
      specifica la modalità di gestione della memoria: paginazione o segmentazione.

    - private final int POLITICA_SCHEDULAZIONE_PROCESSI:
      specifica la politica di schedulazione dei processi(vedi Analisi dei Requisiti).

- private final int DIMENSIONE_RAM:
  specifica la dimensione della memoria centrale RAM espressa in KB.

- private final int DIMENSIONE_SWAP:
  specifica la dimensione dell'area di Swap espressa in KB.

- private final int TEMPO_CONTEXT_SWITCH:
  specifica il tempo di context-switch espresso in millisecondi.

- private final int TEMPO_ACCESSO_DISCO:
  specifica il tempo di accesso al disco espresso in millisecondi.

- private final int DIMENSIONE_PAGINA:
  specifica la dimensione di una pagina espressa in KB.

- private final LinkedList<Processo> LISTA_PROCESSI:
  lista dei processi da usare nella simulazione.

- **Costruttori:**
  - public ConfigurazioneIniziale(int bandaBusDati, int politicaGestioneMemoria, int modalitaGestioneMemoria, int politicaSchedulazione, int dimRAM, int dimSwap, int tempoContextSwitch, int tempoAccessoDisco, int dimPagina, LinkedList<Processo> listaProcessi)
    throws EccezioneConfigurazioneNonValida:
    Costruttore della classe. Inizializza tutti i campi dati controllando che i valori inseriti siano corretti e ragionevoli.

- **Metodi:**
  - public int getBandaBusDati():
    ritorna la banda del bus-dati espressa in KB.

  - public int getDimensionePagina():
    ritorna la dimensione di una pagina espressa in KB.

  - public int getDimensioneRAM():
    ritorna la dimensione della RAM espressa in KB.

  - public int getDimensioneSwap():
    ritorna la dimensione dell'area di Swap espressa in KB.

- public int getModalitaGestioneMemoria():
  ritorna la modalità di gestione della memoria.

- public int getPoliticaGestioneMemoria():
  ritorna la politica di gestione della memoria.

- public int getPoliticaSchedulazioneProcessi():
  ritorna la politica di schedulazione dei processi.

- public int getTempoAccessoDisco():
  ritorna il tempo di accesso al disco espresso in millisecondi.

- public int getTempoContextSwitch():
  ritorna il tempo di context-switch espresso in millisecondi.

- public LinkedList<Processo> getListaProcessi():
  ritorna la lista dei processi che saranno oggetto della simulazione.

#### 3.4.4.2 EccezioneConfigurazioneNonValida

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; estende la classe Exception. Rappresenta un'eccezione lanciata nel momento in cui viene creata una ConfigurazioneIniziale non valida.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalla classe ConfigurazioneIniziale.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    Nessuno.

  - **Costruttori:**
    - Quello di default.
  - **Metodi:**
    Nessuno.

#### 3.4.4.3 Id

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe che si occupa di generare gli identificatori univoci da asseganre ai processi.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da Processo per generare il proprio id.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private static int counter:
      Campo dati statico che conta le istanze degli oggetti di tipo Id.

  - **Costruttori:**
    - public Id():
      Costruttore unico della classe.

  - **Metodi:**
    - public static void resetCounter()
      Metodo che si occupa di resettare il campo dati counter in modo da far ripartire l'assegnazione degli id da zero.

    - public static int returnNewIdi()
      Metodo che si occupa di ritornare un intero rappresentante un nuovo id.

#### 3.4.4.4 Processo

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe Processo rappresenta la classe base per tutti i processi. Essa contiene in sé tutti i campi dati e metodi comuni ad ogni tipo di processo.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getIstanteRichiesta della classe interna Accesso. Utilizza la

classe FrameMemoria per impostare le richieste di accesso alla memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da ConfigurazioneIniziale, da Processore e infine da Scheduler, come lista di processi in arrivo.
  Viene usata da PCB come campo dati.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private java.utils.ArrayLIst<Accesso> accessi:
      Campo dati che contiene la lista degli accessi a FrameMemoria che un processo andrà a richiedere durante il suo ciclo di vita.

    - private int id:
      Campo dati che identifica univocamente ogni istanza della classe. Da inizializzare con un'apposita chiamata ad un metodo della classe Id.

    - private java.lang.String nome:
      Campo dati contenente il nome associato al processo

    - private int tempoArrivo:
      Campo dati contenente l'istante in cui un processo diventa attivo, ciò è l'istante a partire dal quale un processo può concorrere per andare in esecuzione.

    - private int tempoEsecuzione:
      Campo dati che contiene il tempo totale d'esecuzione di un processo.

  - **Costruttori:**
    - public Processo(java.lang.String nome, int tarrivo, int tesecuzione ):
      Costrutture di un'istanza della classe Processo. Necessita del nome, del tempo di arrivo e di quello di esecuzione.

  - **Metodi:**
    - public boolean equals(java.lang.Object processo)
      Metodo che esegue l'uguaglianza fra due processi

    - public java.lang.ArrayList getAccessi()
      Metodo che ritorna la lista degli accessi ordinata per istante di

richiesta crescente.

- public int getId()
  Metodo che permette di visualizzare l'id di un oggetto che istanzia la classe

- public java.lang.String getNome()
  Metodo che ritorna il nome associato ad un processo

- public int getTempoArrivo():
  Metodo che ritorna l'istante di attivazione di un processo.

- public int getTempoEsecuzione()
  Metodo che ritorna il tempo totale d'esecuzione di un processo.

- public boolean richiestaFrameMemoria(FrameMemoria frame, int richiesta)
  Metodo che si occupa di aggiungere una richiesta d'accesso ad un FrameMemoria nel campo dati accessi.

- public java.lang.String toString()
  Metodo che converte a stringa un processo ritornandone il nome.

#### 3.4.4.5 ProcessoConPriorita

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe ProcessoConPriorita estende la classe Processo e rappresenta i processi che, oltre ai dati comuni ad ogni processo ed ereditati dunque dalla classe Processo, sono caratterizzati dall'avere una certa priorità, la quale non si esclude possa anche cambiare nel corso dell'evoluzione della simulazione.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getIstanteRichiesta della classe interna Accesso. Utilizza la classe FrameMemoria per impostare le richieste di accesso alla memoria.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata da tutte le implementazioni di PoliticaOrdinamentoProcessi con priorità.

- **Attività svolte e dati trattati:**

- **Campi dati:**
  - private int priorita:
    Campo dati contenente la priorità associata al processo.

- **Costruttori:**
  - public Processo(java.lang.String nome, int tarrivo, int tesecuzione, int priorita):
    L'unico costruttore. Invoca il costruttore di Processo e inoltre imposta il campo dati priorita.

- **Metodi**:
  - public int getPriorita()
    Metodo che ritorna il valore di priorità associato al processo.

  - public void setPriorita(int priorita)
    Metodo che si occupa di definire la priorità da associare al processo.

### 3.4.5 logic.simulazione

#### 3.4.5.1 Simulazione

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; crea e avvia un processore che crea tutti gli istanti della simulazione. Tali istanti vengono mantenuti internamente a questa classe.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo creaSimulazione() della classe Processore per creare gli istanti della simulazione.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalla classe Player.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private Processore proc:
      istanza di processore che si occupa della generazione degli istanti.

    - private LinkedList<Istante> listaIstanti:
      lista di istanti generata dal processore.

- private ConfigurazioneIniziale conf:
  parametri di configurazione per il sistema di simulazione.

- **Costruttori:**
  - public Simulazione(ConfigurazioneIniziale conf):
    costruttore della classe, che riceve come parametro la configurazione iniziale del sistema

- **Metodi:**
  - public LinkedList<Istante> crea():
    da il via alla creazione della simulazione: il processore può cominciare ad interagire con lo scheduler e il gestore della memoria.

  - public int numeroIstanti():
    ritorna il numero degli istanti che compongono la simulazione.

#### 3.4.5.2 Player

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; scorre gli istanti che compongono una simulazione, in avanti o in dietro nel tempo.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo crea() della classe Simulazione per creare la simulazione con i relativi istanti. Vengono inoltre utilizzati i metodi get della classe Istante, per identificare eventuali eventi rilevanti nel corso della simulazione. Vengono utilizzati i metodi AggiornaStatistiche e AzzeraStatistiche della classe interna Statistiche.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalla GUI per lo scorrimento della simulazione.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private Simulazione simulazioneEseguita:
      la simulazione contenente gli istanti da scorrere con il Player.

    - private LinkedList<Istante> listaIstanti:
      la lista degli istanti generati dalla simulazione.

- private int indiceElementoCorrente:
  l'indice dell'istante corrente; utilizzato per ottimizzare lo scorrimento degli istanti.

- private ListIterator<Istante> istanteCorrente:
  Iteratore che punta all'istante corrente; viene utilizzato per scorrere la lista.

- public enum Evento{FAULT, SWITCH, FULL_RAM, FULL_SWAP, END_PROC, NEW_PROC}:
  Enumerazione per la definizione degli eventi significativi che possono essere ricercati nella lista di istanti.

- public static Statistiche stat:
  Istanza della classe Statistiche.

- **Costruttori:**
  - public Player(ConfigurazioneIniziale conf):
    costruttore della classe, che riceve come parametro la configurazione iniziale del sistema.

- **Metodi:**
  - public boolean caricaSimulazione():
    carica la simulazione e inizializza l'iteratore per lo scorrimento della simulazione; ritorna un booleano che comunica lo stato di successo o fallimento dell'operazione.

  - public Istante istantePrecedente():
    ritorna l'istante precedente a quello attuale. Se viene ritornato un riferimento nullo, significa che non ci sono istanti precedente.

  - public Istante istanteSuccessivo():
    ritorna l'istante successivo a quello attuale. Se viene ritornato un puntatore nullo, significa che non ci sono istanti successivi.

  - public LinkedList<Istante> precedenteIstanteSignificativo (Evento e):
    ritorna l'istante significativo precedente all'istante corrente. Un istante significativo, corrisponde ad uno o più eventi del tipo: fault in memoria principale RAM, context-switch, riempimento della memoria centrale,

riempimento dell'area di Swap, terminazione di un processo, arrivo di uno o più processi.

- public LinkedList<Istante> prossimoIstanteSignificativo (Evento e):
  ritorna l'istante significativo successivo all'istante corrente. Un istante significativo, corrisponde ad uno o più eventi del tipo: fault in memoria principale RAM, context-switch, riempimento della memoria centrale, riempimento dell'area di Swap, terminazione di un processo, arrivo di uno o più processi.

- public Istante primoIstante():
  Ritorna il primo istante della simulazione.

- public LinkedList<Istante> ultimoIstante():
  Ritorna una lista di istanti che porta all'ultimo istante della simulazione.

- public int numeroIstanti():
  Ritorna il numero di istanti che compongono la simulazione.

- public Statistiche getStatistiche():
  Ritorna un riferimento all'oggetto interno stat in modo che dall'esterno siano accessibili i metodi della classe Statistiche.

#### 3.4.5.3 Statistiche

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta, interna alla classe Player; raccoglie dati statistici nel range di istanti che va dall'inizio della simulazione all'istante corrente.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalla classe Player per aggiornare le statistiche.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private int utilizzoRAM:
      la quantità in KB di RAM utilizzata.

- private int utilizzoSwap:
  la quantità in KB di Swap utilizzata.

- private int numeroFault:
  numero di Fault avvenuti in memoria.

- private int numeroIstantiRimanenti:
  numero di istanti rimanenti alla fine della simulazione.

- **Costruttori:**
  - Statistiche():
    Costruttore della classe. Inizializza tutti i campi a zero.

- **Metodi:**
  - public int getUtilizzoRAM():
    ritorna la quantità in KB di RAM utilizzata.

  - public int getUtilizzoSwap():
    ritorna la quantità in KB di Swap utilizzata.

  - public int getNumeroFault():
    ritorna il numero di fault avvenuti in memoria.

  - public int getNumeroIstantiRimanenti():
    ritorna il numero di istanti rimanenti alla fine della simulazione.

  - private void AggiornaOccupazioni(Istante nuovoIstante, boolean avanti):
    aggiorna i campi utilizzoRAM e utilizzoSwap.

  - void AggiornaStatistiche(Istante nuovoIstante, boolean avanti ):
    aggiorna le statistiche specificando il nuovo istante, e se quest'ultimo è precedente o successivo a quello corrente.

  - void AggiornaStatistiche(LinkedList<Istante> listaNuoviIstanti, boolean avanti):
    aggiorna le statistiche specificando una lista di Istante, e se quest'ultimi sono precedenti o successivi all'istante corrente.

- void AzzeraStatistiche():
  azzera le statistiche.

#### 3.4.5.4 Istante

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe pubblica e concreta; rappresenta una singola unità di tempo della simulazione. Al suo interno le informazioni sono rappresentate in modo differenziale; ciò significa che ogni istante rappresenta solo le differenze rispetto all'istante precedente.

- **Relazioni d'uso con altre componenti:**
  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dalla classe Player la quale interroga i campi dato per scoprire eventi significativi.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private PCB processoInEsecuzione:
      il processo in esecuzione nel'istante corrente.
    - private PCB processoPrecedenteTerminato:
      il processo terminato nell'istante corrente, cioè che ha eseguito il suo ultimo quanto di tempo nell'istante precedente.
    - private boolean nuovoProcesso:
      indica se sono arrivati nuovi processi nella coda dei pronti.
    - private int numeroFault:
      numero di fault generati nell'istante corrente.
    - private LinkedList<Azione> cambiamentiInMemoria:
      i cambiamenti avvenuti in memoria (RAM e Swap).
    - private boolean full_RAM:
      indica se la memoria RAM si è riempita a tal punto da non poter caricare nuove pagine/segmenti.

- private boolean full_Swap:
  indica se l'area di Swap ha raggiunto la sua capacità massima.

- private Vector<FrameMemoria> statoRAM
  indica lo stato della RAM

- private Vector<FrameMemoria> statoSwap
  indica lo stato dello Swap

- **Costruttori:**
  - public Istante(PCB inEsecuzione, PCB terminato, boolean nuovoProcesso, int fault,LinkedList<Azione> memoria, boolean full_RAM, boolean full_Swap):
    costruttore della classe. Inizializza tutti i campi dati precedentemente descritti.

- **Metodi:**
  - public PCB getProcessoInEsecuzione():
    ritorna il PCB del processo in esecuzione nell'istante corrente.

  - public PCB getProcessoPrecedenteTerminato():
    ritorna il PCB del processo terminato nell'istante corrente.

  - public boolean getNuovoProcesso():
    ritorna un booleano che indica se nell'istante corrente sono arrivati nuovi processi nella coda dei pronti.

  - public int getFault():
    ritorna il numero di fault generati in memoria nell'istante corrente.

  - public LinkedList<Azione> getCambiamentiInMemoria():
    ritorna una lista di Azione, che indica ogni operazione elementare avvenuta in memoria. Per maggiori dettagli vedere la documentazione della classe Azione.

  - public boolean getFull_RAM():
    ritorna un booleano che indica se la RAM è stata riempita a tal punto da non permettere l'inserimento di nuove pagine/segmenti, fino a che una parte di essa non sarà liberata.

- public boolean getFull_Swap():
  ritorna un booleano che indica se nell'istante corrente l'area di Swap ha raggiunto la sua capacità massima.

- public Vector<FrameMemoria> getStatoRAM()
  Ritorna lo stato della RAM

- public Vector<FrameMemoria> getStatoSwap()
  Ritorna lo stato dello Swap

### 3.4.6 logic.schedulazione

#### 3.4.6.1 PoliticaOrdinamentoProcessi

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Una Politica di Ordinamento dei Processi ha il compito di ordinare i processi che sono attivi in un sistema, scegliendo quale di questi deve eseguire ad ogni istante. Una politica di ordinamento viene specializzata a seconda dei sistemi in cui questa verrà applicata. Elemento comune tra le varie politiche di ordinamento, è quello di dover mantenere i processi pronti all'interno di una struttura dati.

- **Relazioni d'uso con altre componenti:**
  Nessuna

- **Interfacce e relazioni di uso da altre componenti:**
  Nessuna (vengono utilizzate le sue implementazioni).

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    Nessuno

  - **Costruttori:**
    Nessuno

  - **Metodi:**
    - public void esegui()
      Metodo che simula l'esecuzione del processo che sta detenendo la

CPU. La simulazione dell'esecuzione avviene aumentando il numero di istanti eseguiti di tale processo di un certo valore passato come parametro, valore che viene calcolato dallo scheduler.

- public java.util.ArrayList getCodaPronti()

  Metodo che ritorna un ArrayList contenente tutti i riferimenti a Processo a cui sono associati i PCB contenuti nella coda dei pronti

- public PCB estrai()

  Rimuove il primo elemento dalla struttura dati definita nella classe che lo implementerà.

- public void setScheduler(Scheduler scheduler)

  Metodo che imposta il campo dato di tipo Scheduler che dovrà essere previsto da ogni politica di ordinamento concreta che implementi direttamente questa interfaccia.

- public void inserisci(PCB pcb)

  Inserisce il PCB passato per riferimento nella struttura dati della classe che implementerà questa interfaccia.

- public int size()

  Ritorna il numero di PCB correnti presenti nella struttura dati definita nella classe che andrà ad implementarlo.

#### 3.4.6.2 ConQuanti

- **Tipo, obiettivo e funzione della classe/interfaccia:**

  Classe astratta che implementa l'interfaccia PoliticaOrdinamentoProcessi. Lo scopo di questa classe è quello di definire una ulteriore caratteristica del comportamento di una politica di ordinamento concreta. Una politica con quanti stabilisce un intervallo di tempo (detto timeslice) durante il quale un processo può eseguire. Alla scadenza del quanto di tempo, se il processo non ha finito la sua esecuzione esso potrà essere re-inserito nella coda dei processi pronti oppure continuare con la sua esecuzione a seconda di cosa le classi concrete che implementano la presente prevedranno di fare.

- **Relazioni d'uso con altre componenti:**

  Nessuna.

- **Interfacce e relazioni di uso da altre componenti:**
  Nessuna.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private int contatore:
      Campo dato di tipo intero che indica per il processo attualmente in esecuzione quante unità di tempo del quanto a sua disposizione sono state sfruttate.

  - **Costruttori:**
    - public ConQuanti(int timeSlice):
      costruttore della classe, che riceve come parametro il valore del timeslice della politica.

  - **Metodi:**
    - public int getContatore():
      Ritorna il valore del contatore che si occupa di quante unità di tempo, rispetto al timeSlice sono state sfruttate dal processo in esecuzione.

    - public int getTimeSlice():
      Ritorna il valore del campo dato timeSlice.

    - public void reset():
      Azzera il valore del contatore dei quanti di tempo sfruttati dal PCB in esecuzione.

    - public void setContatore(int contatore):
      Setta il valore del contatore.

    - public void setTimeSlice(int timeSlice):
      Setta il valore di timeslice.

#### 3.4.6.3 FCFS

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  La classe FCFS implementa l'interfaccia PoliticaOrdinamentoProcessi e rappresenta il funzionamento della politica di ordinamento First Come First Served, tipica dei sistemi Batch. Questo tipo di politica ordina la coda dei processi pronti secondo il loro tempo d'arrivo e li fa eseguire uno alla volta fino al completamento. La classe possiede quindi una struttura dati di

tipo FIFO, contenente PCB, sulla quale vengono fatte le operazioni di inserimento e di estrazione. Dunque, l'estrazione del PCB da mandare in esecuzione sarà fatto in testa alla struttura mentre ogni nuovo PCB che entra nello stato di pronto verrà inserito in coda.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo IncrementaTempoScheduler dell'oggetto scheduler per simulare l'esecuzione da parte del metodo che possiede la CPU, nel metodo esegui.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected java.utils.LinkedList codaPronti:
      Coda dei processi pronti, si comporta come una coda FIFO.

    - protected Scheduler scheduler:
      Riferimento allo scheduler.

  - **Costruttori:**
    - public FCFS():
      Costruttore che si occupa di inizializzare la struttura dati utilizzata per rappresentare la coda dei processi pronti per l'esecuzione.

  - **Metodi:**
    - public void esegui():
      Simula l'esecuzione del processo che “possiede” la CPU.

    - public java.util.ArrayList getCodaPronti()
      Metodo che ritorna un ArrayList contenente tutti i riferimenti a Processo a cui sono associati i PCB contenuti nella coda dei pronti

    - public PCB estrai()
      Rimuove il primo elemento dalla coda dei pronti.

    - public void setScheduler(Scheduler scheduler)
      Metodo che imposta il campo scheduler.

- public void inserisci(PCB pcb)
  Inserisce il PCB passato per riferimento nella coda dei pronti

- public int size()
  Ritorna il numero di PCB correnti presenti nella coda dei pronti.

#### 3.4.6.4 PCB

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe che rappresenta un process control block, ovvero un particolare oggetto che indicheremo con l'acronimo PCB e che si occupa di mantenere memorizzati tutti i dati di un processo che evolvono col trascorrere della simulazione. Istanze di questa classe perciò si occupano di memorizzare i dati dinamici relativi a dei processi. Ogni processo sarà associato ad un'istanza di questa classe e ogni istanza di questa classe sarà associata ad uno ed un solo processo. Dunque dall'istanza di PCB potrà identificare in modo univoco il processo associato, risalendo ai dati dinamici che lo caratterizzano.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getTempoEsecuzione dell'oggetto processo per determinare gli istanti da eseguire, al momento della creazione.
  Utilizza il metodo equals dell'oggetto processo per confrontare due processi rappresentati da due PCB.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene utilizzata da ogni implementazione dell'interfaccia PoliticaOrdinamentoProcessi per mantenere la lista dei PCB rappresentanti i processi nello stato di ponto.
  Viene utilizzata dalla classe Scheduler che crea ogni istanza di PCB a partire da un processo che passa dallo stato “in arrivo” a quello “terminato”. Lo Scheduler inoltre mantiene un riferimento al processo attualmente in esecuzione e ad una lista di quelli terminati.
  Viene utilizzata dalla classe Processore, che riceve dallo Scheduler un riferimento al PCB in esecuzione, ne estrae i FrameMemoria necessari e controlla che il PCB precedentemente in esecuzione non sia terminato.
  Viene utilizzata dalla classe Istante, di cui ogni istanza memorizza il PCB in esecuzione e quello appena terminato in un dato istante di tempo della simulazione.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - private int istantiDaEseguire:
      Campo dati contenente in ogni momento il numero totale di istanti d'esecuzione che mancano al processo associato al PCB per terminare la sua esecuzione.

    - private int istantiEseguiti:
      Campo dati contenente per ogni unità di tempo il numero di istanti per cui il processo associato al PCB ha eseguito.

    - private Processo processo:
      Campo dati che si occupa di mantenere un riferimento al processo a cui il PCB sarà associato.

  - **Costruttori:**
    - public PCB(Processo processo):
      Costruttore si occupa di creare un'istanza della classe PCB. In esso vengono inizializzati i campi dati processo, col processo da associare al PCB, e istantiDaEseguire, col valore totale degli istanti d'esecuzione del processo. Gli altri campi della classe PCB verranno definiti con inizializzazioni automatiche.

  - **Metodi:**
    - public boolean equals(java.lang.Object o):
      Metodo che permette di verificare se due oggetti di tipo PCB sono uguali. Due oggetti PCB sono uguali solo se si riferiscono allo stesso processo. Dunque il metodo al suo interno reinvoca il metodo equals definito per la classe Processo.

    - public int getIstantiDaEseguire()
      Metodo che ritorna il numero di istanti per cui il processo deve ancora eseguire prima di terminare.

    - public int getIstantiEseguiti()
      Ritorna il numero di istanti eseguiti dal processo associato.

    - public Processo getRifProcesso()
      Metodo che permette d'accedere al valore del campo dati processo.

- public void incIstantiEseguiti()
  Metodo che incrementa gli istantiEseguiti e decrementa gli istantiDaEseguire di un'unità di tempo.

#### 3.4.6.5 Priorita

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Questa classe implementa la politica di ordinamento con priorità, politica che richiede che i PCB da gestire siano associati ad oggetti di tipo ProcessoConPriorita, in quanto l'ordinamento della coda dei pronti si basa sulla priorità. La coda dei PCB pronti è ordinata per priorità decrescente e viene estratto il PCB con priorità maggiore.

- **Relazioni d'uso con altre componenti:**
  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Interfacce e relazioni di uso da altre componenti:**
  Nessuna

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected java.utils.LinkedList codaPronti:
      Coda dei processi pronti, si comporta come una coda FIFO.

    - protected Scheduler scheduler:
      Riferimento allo scheduler.

  - **Costruttori:**
    - public Priorita():
      Costruttore che si occupa di inizializzare la struttura dati utilizzata per rappresentare la coda dei processi pronti per l'esecuzione.

  - **Metodi:**
    - public void esegui():
      Simula l'esecuzione del processo che “possiede” la CPU.

    - public java.util.ArrayList getCodaPronti()
      Metodo che ritorna un ArrayList contenente tutti i riferimenti a

Processo a cui sono associati i PCB contenuti nella coda dei pronti

- public PCB estrai()
  Estrae il PCB che deve eseguire. In particolare questo è il primo della coda, ossia il PCB con priorità maggiore.

- public void setScheduler(Scheduler scheduler)
  Metodo che imposta il campo scheduler.

- public void inserisci(PCB pcb)
  Metodo che effettua l'inserimento di un PCB nella coda dei processi pronti. L'inserimento mantiene la coda ordinata per priorità decrescente.

- public int size()
  Ritorna il numero di PCB correnti presenti nella coda dei pronti.

#### 3.4.6.6 RR

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe concreta, estende la classe astratta ConQuanti e implementa l'interfaccia PoliticaOrdinamentoProcessi. Questa classe ha il compito di simulare una politica di ordinamento per sistemi interattivi. La politica Round Robin, è una politica con ordinamento a quanti, ovvero ogni processo esegue per al più un quanto di tempo alla volta. La struttura dati su cui vengono mantenuti i processi pronti diventa quindi una lista circolare di processi. L'inserimento viene sempre fatto in coda a questa lista, mentre l'estrazione dalla testa.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo IncrementaTempoScheduler dell'oggetto scheduler per simulare l'esecuzione da parte del metodo che possiede la CPU, nel metodo esegui.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Attività svolte e dati trattati:**
  - **Campi dati:**

- protected java.utils.LinkedList codaPronti:
  Coda dei processi pronti, si comporta come una coda FIFO.

- protected Scheduler scheduler:
  Riferimento allo scheduler.

- private PCB ultimoEseguito:
  Riferimento dell'ultimo PCB mandato in esecuzione. Servirà nel metodo esegui e, nel caso ultimoEseguito e il PCB in esecuzione fossero differenti, deve essere resettato il contatore.

- **Costruttori:**
  - public RR():
    Costruttore di default. Richiama il costruttore ad un parametro, con valore 3. Assegna di default un valore di quanto uguale a 3.

  - public RR(int timeSlice):
    Costruttore che riceve come parametro un intero rappresentante il valore di time slice scelto.

- **Metodi:**
  - public void esegui():
    Simula l'esecuzione del processo che “possiede” la CPU.

  - public java.util.ArrayList getCodaPronti()
    Metodo che ritorna un ArrayList contenente tutti i riferimenti a Processo a cui sono associati i PCB contenuti nella coda dei pronti

  - public PCB estrai()
    Rimuove il primo elemento dalla coda dei pronti.

  - public void setScheduler(Scheduler scheduler)
    Metodo che imposta il campo scheduler.

  - public void inserisci(PCB pcb)
    Inserisce il PCB passato per riferimento nella coda dei pronti

  - public int size()
    Ritorna il numero di PCB correnti presenti nella coda dei pronti.

### 3.4.6.7 RRConPriorita

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe concreta, estende la classe astratta ConQuanti, e implementa l'interfaccia PoliticaOrdinamentoProcessi. Questa classe ha il compito di simulare una politica di ordinamento Round Robin con priorità. Questa è una politica con ordinamento a quanti, ovvero ogni processo esegue per al più un quanto di tempo alla volta. La particolarità di questa politica, a differenza di quella Round Robin, è che i processi vengono mantenuti in una struttura dati ordinata per priorità. Ogni processo coinvolto in questa politica, ha infatti un valore di priorità proprio, che determina in quale lista della struttura dati questo deve essere inserito. L'estrazione di un PCB associato ad un processo, procederà cercando quello con priorità più alta. Al termine della sua esecuzione il PCB verrà rimesso in coda alla lista da cui Ã¨ stato estratto. La politica non prevede prerilascio per priorità, quindi se durante l'esecuzione di un processo con priorità bassa, ne arrivasse uno con priorità più alta, il primo rimarrebbe in esecuzione fino al termine del quanto di tempo a sua disposizione, mentre l'altro verrebbe inserito nella coda dei pronti. Non capiterà che un PCB con priorità bassa vada in esecuzione prima di un PCB con priorità alta, già presente nella coda dei pronti.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getPriorita della classe ProcessoConPriorita.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected java.utils.LinkedList codaPronti:
      Coda dei processi pronti, si comporta come una coda FIFO.

    - protected Scheduler scheduler:
      Riferimento allo scheduler.

    - private PCB ultimoEseguito:
      Riferimento dell'ultimo PCB mandato in esecuzione. Servirà nel metodo esegui e, nel caso ultimoEseguito e il PCB in esecuzione fossero differenti, deve essere resettato il contatore.

- **Costruttori:**
  - public RRConPriorita():
    Costruttore di default. Richiama il costruttore ad un parametro, con valore 3. Assegna di default un valore di quanto uguale a 3.

  - public RRConPriorita(int timeSlice):
    Costruttore che riceve come parametro un intero rappresentante il valore di time slice scelto.

- **Metodi:**
  - public void esegui():
    Simula l'esecuzione del processo che “possiede” la CPU.

  - public java.util.ArrayList getCodaPronti()

    Metodo che ritorna un ArrayList contenente tutti i riferimenti a Processo a cui sono associati i PCB contenuti nella coda dei pronti

  - public PCB estrai()
    Estrae e ritorna il PCB del processo che deve andare in esecuzione. L'estrazione viene fatta cercando il PCB con priorità maggiore all'interno della codaPronti.

  - public void setScheduler(Scheduler scheduler)
    Metodo che imposta il campo scheduler.

  - public void inserisci(PCB pcb)
    Inserisce il PCB passato per riferimento nella coda dei pronti

  - public int size()
    Ritorna il numero di PCB correnti presenti nella coda dei pronti.

#### 3.4.6.8 RRConPrioritaConPrerilascio

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe concreta che estende la classe concreta RRConPriorita. Questa classe ha il compito di simulare una politica di ordinamento per sistemi interattivi, più precisamente la politica Round Robin con priorità e con prerilascio per priorità. Questa è una politica con ordinamento a quanti, ovvero ogni processo esegue per al più un quanto di tempo alla volta. La

particolarità di questa politica, a differenza di quella Round Robin, è che i processi vengono mantenuti in una struttura dati ordinata per priorità. Ogni processo coinvolto in questa politica, ha infatti un valore di priorità proprio, che determina in quale lista della struttura dati questo deve essere inserito. L'estrazione di un PCB associato ad un processo, procederà cercando quello con priorità più alta. Al termine della sua esecuzione il PCB verrà rimesso in coda alla lista da cui è stato estratto. La politica, a differenza di quella da cui deriva, prevede prerilascio per priorità, quindi nel caso stesse eseguendo un processo con priorità bassa, e ne arrivasse uno con priorità più alta, il primo verrebbe rimesso nella coda dei pronti, e mandato in esecuzione quello appena arrivato.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getPriorita della classe ProcessoConPriorita.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    Nessuno

  - **Costruttori:**
    - public RRConPrioritaConPrerilascio():
      Costruttore di default. Richiama il costruttore ad un parametro, con valore 3. Assegna di default un valore di quanto uguale a 3.

    - public RRConPrioritaConPrerilascio(int timeSlice):
      Costruttore che riceve come parametro un intero rappresentante il valore di time slice scelto.

  - **Metodi**:
    - public void inserisci(PCB pcb)
      Metodo per inserire un nuovo PCB pronto all'interno della struttura dati. Prima di inserire il PCB del processo pronto nella codaPronti, viene controllato se la priorità di quest'ultimo è maggiore del PCB del processo che sta eseguendo. In questo caso il PCB del processo in esecuzione viene reinserito nella coda dei pronti e viene schedulato quello appena arrivato. Altrimenti il PCB passato come parametro

viene inserito come avviene nello stesso metodo della classe RRConPriorita.

- public PCB minore(PCB pronto, PCB inEsecuzione)
  Metodo di confronto tra due PCB, vengono confrontate le priorità dei PCB passati come parametri.

#### 3.4.6.9 SJF

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Classe concreta. Questa classe rappresenta la politica di ordinamento Shortest Job First, che viene usualmente applicata nei sistemi Batch. Essa richiede la conoscenza dei tempi di esecuzione infatti viene eseguito sempre il lavoro più breve. Ovviamente, non è equo con i lavori non presenti all'inizio, a differenza della classe che estenderà questa politica (SRTF). La struttura dati che conterrà i processi pronti sarà quindi ordinata per tempi di esecuzione crescenti.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getIstantiDaEseguire della classe PCB.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - protected java.utils.LinkedList codaPronti:
      Coda dei processi pronti, è ordinata per tempo di esecuzione crescente.
    - protected Scheduler scheduler:
      Riferimento allo scheduler.
  - **Costruttori:**
    - public SJF():
      Costruttore di default. Richiama il costruttore ad un parametro, con valore 3. Assegna di default un valore di quanto uguale a 3.
  - **Metodi:**
    - public void esegui():

Simula l'esecuzione del processo che “possiede” la CPU.

- public java.util.ArrayList getCodaPronti()

  Metodo che ritorna un ArrayList contenente tutti i riferimenti a Processo a cui sono associati i PCB contenuti nella coda dei pronti

- public PCB estrai()

  Rimuove il primo elemento dalla coda dei pronti.

- public void setScheduler(Scheduler scheduler)

  Metodo che imposta il campo scheduler.

- public void inserisci(PCB pcb)

  Inserisce un nuovo PCB nella coda dei processi pronti. Viene effettuato mantenendo la struttura dati ordinata per tempo di esecuzione crescente.

- public int size()

  Ritorna il numero di PCB correnti presenti nella coda dei pronti.

#### 3.4.6.10 SRTN

- **Tipo, obiettivo e funzione della classe/interfaccia:**

  Classe concreta che estende la classe SJF. La classe SRTN implementa la politica di ordinamento Shortest Remaining Time Next, ovvero una politica con prerilascio per sistemi Batch. Eredita i metodi esegui(), estrai(), size() e getCodaPronti() dalla classe superiore, mentre implementa il metodo minore(PCB a, PCB b) Questa politica di ordinamento ha un comportamento simile alla politica SJF, ma applica il prerilascio quando arriva un PCB con tempo di esecuzione minore rispetto a quello che sta eseguendo. I PCB vengono mantenuti ordinati rispetto al tempo di esecuzione in maniera crescente, su una struttura dati. L'estrazione invece viene fatta dalla testa della struttura dati, proprio perché questa viene mantenuta ordinata.

- **Relazioni d'uso con altre componenti:**

  Utilizza il metodo getIstantiDaEseguire della classe PCB.

- **Interfacce e relazioni di uso da altre componenti:**

  Viene usata dallo Scheduler nel caso sia la PoliticaOrdinamentoProcessi scelta.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    Nessuno.

  - **Costruttori:**
    - public SRTN():
      Costruttore, richiama i costruttori della classe superiore (SJF), e crea la struttura dati per implementare la politica di ordinamento SRTN.

  - **Metodi:**
    - public PCB minore(PCB pronto, PCB inEsecuzione)
      Metodo di confronto tra due PCB, vengono confrontati i tempi di esecuzione dei PCB passati come parametri.

    - public void inserisci(PCB pcb)
      Inserisce un nuovo PCB pronto all'interno della struttura dati. Prima di inserire il PCB del processo pronto nella codaPronti, viene controllato se il tempo di esecuzione di quest'ultimo è minore del PCB del processo che sta eseguendo. In questo caso il PCB del processo in esecuzione viene reinserito nella coda dei pronti e viene schedulato quello appena arrivato. Altrimenti il PCB passato come parametro viene inserito come avviene nello stesso metodo della classe SJF.

### 3.4.6.11 Scheduler

- **Tipo, obiettivo e funzione della classe/interfaccia:**
  Questa classe implementa i meccanismi necessari per realizzare una simulazione discreta di processi in un elaboratore multi-programmato. Lo Scheduler viene interrogato per ogni istante dal Processore, a cui ritorna un riferimento al PCB correntemente in esecuzione, che potrà essere un riferimento nullo nel caso in cui nessun processo abbia il controllo della CPU. L'esecuzione dei vari processi avverrà in maniera specializzata a seconda della politica di ordinamento scelta.

- **Relazioni d'uso con altre componenti:**
  Utilizza il metodo getTempoArrivo della classe Processo.
  Utilizza il metodo inserisci, esegui ed estrai di PoliticaOrdinamentoProcessi.
  Utilizza il metodo incIstantiEseguiti, getIstantiDaEseguire e getRifProcesso di PCB.

- **Interfacce e relazioni di uso da altre componenti:**
  Viene usata dall'interfaccia PoliticaOrdinamentoProcessi, da tutte le sue implementazioni e dal Processore.

- **Attività svolte e dati trattati:**
  - **Campi dati:**
    - package boolean fineSimulazione:
      Indica se la simulazione è giunta al termine o meno.

    - package PCB PCBCorrente:
      Il PCBCorrente rappresenta il PCB che, all'istante definito nella variabile tempoCorrente, è nello stato di esecuzione.

    - package PoliticaOrdinamentoProcessi politicaOrdinamento:
      Questo campo dati specifica la politica di schedulazione che si intende utilizzare per lo scheduler.

    - package java.util.LinkedList processiInArrivo:
      Contiene la lista dei processi da attivare, ordinata per tempo di arrivo.

    - package java.util.ArrayList processiTerminati:
      Questo campo dati contiene la coda dei processi terminati in ordine di terminazione.

    - package PCB PCBCorrente:
      Il PCBCorrente rappresenta il PCB che, all'istante definito nella variabile tempoCorrente, è nello stato di esecuzione.

    - package int tempoCorrente:
      E' il contatore interno dello Scheduler.

    - package int[] tempoEvento:
      Questa struttura serve per memorizzare i tempi dei due eventi che possono ricorrere in questa simulazione di schedulazione: l'arrivo e la terminazione di un processo.

    - package int tempoProssimoEvento:
      Salva il valore ritornato dal metodo tempoProssimoEvento().

- **Costruttori:**
  - public Scheduler(PoliticaOrdinamentoProcessi politicaOrdinamento, java.util.LinkedList processiInArrivo):
    L'unico costruttore della classe.

- **Metodi:**
  - package void attivaProcesso()
    Questo metodo si occupa di attivare un nuovo processo arrivato prendendo il primo della lista dei processi in arrivo, ne crea il PCB e lo inserisce nella lista dei pronti a seconda della politica di schedulazione scelta.

  - public boolean eseguiAttivazione()
    Questo metodo invocato dal Processore si occupa di preparare il prossimo PCB in esecuzione. Ritorna true se alla fine della sua esecuzione non è disponibile un PCB in esecuzione, false altrimenti.

  - public void eseguiIncremento():
    Questo metodo viene invocato dallo scheduler solo nel caso ci sia un PCB che può eseguire e fa progredire la simulazione.

  - public boolean fineSimulazione():
    Questo metodo ritorna lo stato della simulazione.

  - public PCB getPCBCorrente:
    Ritorna il PCB attualmente nello stato di esecuzione.

  - public PoliticaOrdinamentoProcessi getPoliticaOrdinamentoProcessi() :
    Ritorna la politica di ordinamento scelta nel costruttore.

  - package java.util.LinkedList getProcessiInArrivo():
    Ritorna la lista dei processi che devono ancora essere attivati, ordinata per tempo di arrivo.

  - public java.util.ArrayList getProcessiTerminati()
    Ritorna la lista dei processi sono terminati in ordine di terminazione.

  - public int getTempoCorrente():
    Ritorna un intero che rappresenta il numero di istanti di simulazione trascorsi.

- package void incrementaTempo():
  Incrementa di un'unità di tempo il contatore del tempo corrente.

- public void incrementaTempo():
  Questo metodo incrementa il tempo corrente dello scheduler e decrementa automaticamente i tempi dei prossimi eventi di una unità.

- package void rimuoviPCBCorrente() :
  Rimuove il PCBCorrente dalla posizione di esecuzione.

- package int tempoProssimoEvento():
  Questo metodo calcola l'intervallo tra il tempo attuale e il prossimo evento.

- package void terminaPCBCorrente() :
  Metodo invocato quando un PCB ha completato la sua esecuzione.

# 4 Appendice

## 4.1 Tracciamento della relazione componenti-requisiti

| Classe | Requisiti |
|---|---|
| SiGeMv2View | RFO11, RFO12, RFO13, RFO14, RFO16, RFO21, RFO23, RFO24, RNO05, RNO10, RNO11, RFP02 |
| ProcessiJDialog | RFO05, RFO15, RFO17, RFO24, RFD02 |
| ConfigurazioneAmbienteJDialog | RFO04, RFO06, RFO07, RFO15, RFO17, RFO24, RFD02 |
| AssociazioneProcessiJDialog | RFO15, RFO17, RFO24, RFD02 |
| PoliticheJDialog | RFO06, RFO07, RFO08, RFO09, RFO15, RFO17, RFO24, RFD02, RFP01 |
| ViewFrameMemoria | RFD03 |
| ViewStatistiche | RFO13, RFO14 |
| ViewStatoAvanzamentoProcessi | RFO11 |
| GestioneFile | RFO16 |
| A | RFO01, RFO10 |

| | |
|---|---|
| AzionePagina | RFO01, RFO10 |
| AzioneSegmento | RFO02, RFO10 |
| BestFit | RFO02, RFO10 |
| C | RFO01, RFO10 |
| FIFO | RFO01, RFO10 |
| FirstFit | RFO02, RFO10 |
| GestoreMemoriaPaginata | RFO01, RFO08, RFO10 |
| GestoreMemoriaSegmentata | RFO02, RFO09, RFO10 |
| LRU | RFO01, RFO10 |
| NextFit | RFO02, RFO10 |
| NFU | RFO01, RFO10 |
| NRU | RFO01, RFO10 |
| Pagina | RFO01 |
| QuickFit | RFO02, RFO10 |
| RAMPaginata | RFO01, RFO13, RFO03, RFO22 |
| RAMSegmentata | RFO02, RFO13, RFO03 |
| SC | RFO01, RFO10 |
| Segmento | RFO02 |
| SwapPaginata | RFO01, RFO13, RFO03, RF022 |
| SwapSegmentata | RFO02, RFO13, RFO03 |
| WorstFit | RFO02, RFO10 |
| ConfigurazioneIniziale | RFO03, RFO04, RFO05, RFO06, RFO07, RFO08, RFO09, RFO10, RFO22, RFP01 |
| Processo | RFO05 |
| ProcessoConPriorita | RFO05 |
| FCFS | RFP01 |
| Priorita | RFP01 |
| RR | RFP01 |
| RRConPriorita | RFP01 |

| | |
|---|---|
| RRConPrioritaConPrerilascio | RFP01 |
| Scheduler | RFP01 |
| SJF | RFP01 |
| SRTN | RFP01 |
| Istante | RFO14 |
| Player | RFO13, RFO18, RFO19, RFD03 |
| Simulazione | RFO13, RFO18, RFO19, RFD03 |
| Processore | RFP01, RFO02, RFO01 |