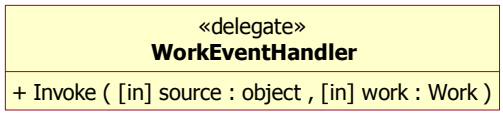
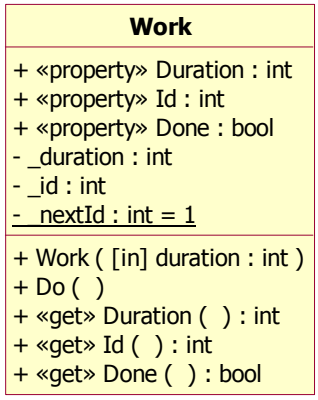
Prova 1

Avvertenze per la consegna. All'inizio di ogni file sorgente inserire in un commento i propri dati: **cognome, nome e numero di matricola**. Al termine dell'esame, **consegnare (inviare) tutti i file sorgente e tutti i file XML che sono stati utilizzati o generati dall'applicazione**.

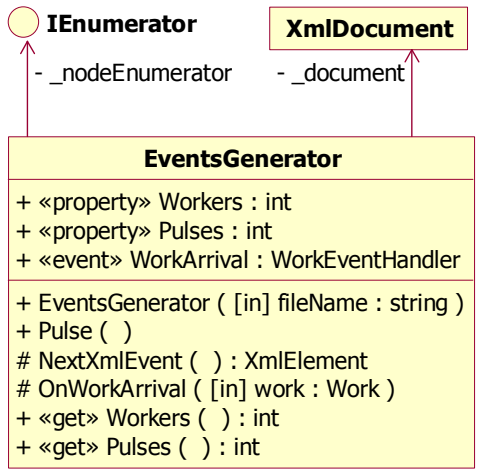
Passo 1 – Creare un nuovo progetto di tipo **`ConsoleApplication`** e di nome **`Prova1`**. Scaricare il file "**`Prova1Start.zip`**" e inserire i file in esso contenuti nello stesso direttorio dei file sorgenti del progetto.



Definire il *delegate* **`WorkEventHandler`**, la cui *signature* è indicata nello schema UML dal metodo **`Invoke`**.



Definire la classe **`Work`** (come da schema UML). La classe **`Work`** descrive un "lavoro" mediante un identificatore univoco (**`_id`**) e una durata corrente (**`_duration`**) in impulsi di clock (sarà chiaro in seguito l'esatto significato). In fase di creazione di una nuova istanza, si utilizzi il *field* statico **`_nextId`** per inizializzare correttamente l'identificatore univoco. Le proprietà **`Id`** e **`Duration`** permettono di accedere ai corrispondenti *field*, mentre la proprietà **`Done`** permette di sapere se il lavoro è terminato (cioè se la durata corrente è zero). Il metodo **`Do`**, infine, simula l'esecuzione di una parte del lavoro, decrementando la durata corrente di 1 unità (si lanci un'eccezione se il metodo viene invocato su un *work* già terminato).

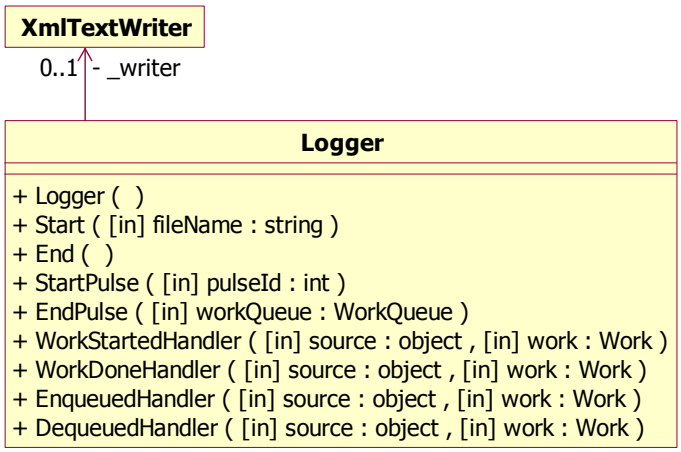


Definire la classe **`EventsGenerator`** (come da schema UML). Il costruttore della classe accetta come argomento il nome di un *file* XML contenente gli eventi da generare (per i dettagli sulla struttura del documento XML, si veda il contenuto del file "**`Input.xml`**"). Il costruttore deve caricare in memoria il documento XML e inizializzare adeguatamente tutti i *field*: in particolare, **`_nodeEnumerator`** deve fare riferimento a un enumeratore sulla lista dei nodi XML figli del nodo "**`Events`**". Le proprietà **`Workers`** e **`Pulses`** devono restituire i valori dei corrispondenti attributi del nodo "**`Events`**". Il metodo **`OnWorkArrival`** deve scatenare l'evento **`WorkArrival`**. Il metodo **`NextXmlEvent`** deve utilizzare **`_nodeEnumerator`** per restituire il successivo nodo "evento" in lista – Attenzione: la scansione deve essere ciclica; pertanto, dopo l'ultimo nodo in lista, è necessario restituire nuovamente il primo nodo e così via (gestire in modo adeguato l'enumeratore). Infine, il metodo **`Pulse`** deve ottenere il successivo nodo "evento" in lista e, solo nel caso di **`<WorkArrival>`**, deve creare una nuova istanza di **`Work`** di durata appropriata e quindi fare scatenare l'evento **`WorkArrival`**.

Nel **`Main`** creare un **`EventsGenerator`**, che utilizzi il file "**`Input.xml`**", agganciare l'evento **`WorkArrival`** al metodo statico **`Program.eventsGenerator_WorkArrival`** e quindi realizzare un ciclo **`for`** che simuli la generazione di una serie di "impulsi" (il numero degli impulsi da generare è dato dal valore dell'attributo "**`pulses`**" del nodo "**`Events`**") e che a ogni iterazione (cioè ad ogni impulso generato) invochi il metodo **`Pulse`** sull'istanza di **`EventsGenerator`**.

Per testare il codice, inserire temporaneamente in **`eventsGenerator_WorkArrival`** una scrittura su console e lanciare l'applicazione: devono venire visualizzati tutti e solo gli eventi **`WorkArrival`** e le istanze di **`Work`** devono avere identificatori univoci e crescenti.

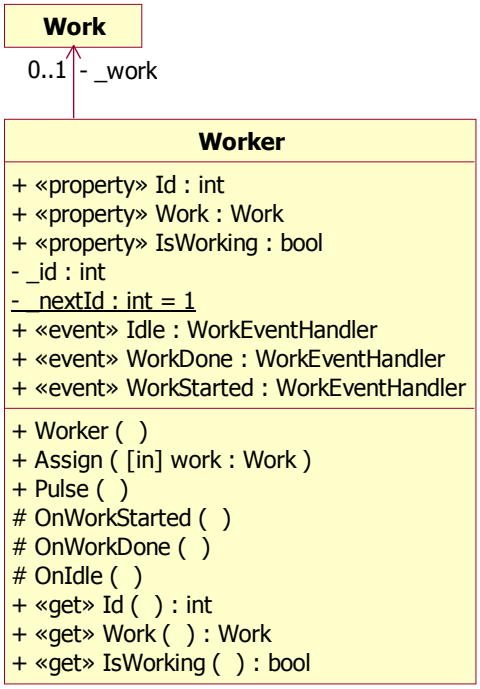
Prova 1



Passo 2 – Definire la classe **`Logger`** (come da schema UML) il cui compito è quello di riportare su un file XML tutte le operazioni effettuate durante un *run* dell'applicazione. La struttura del documento XML da generare è riportata nel file "**`OutputAtteso.xml`**". Il metodo **`Start`** accetta come argomento il nome del file XML di output, crea in modo adeguato un **`XmlTextWriter`** e apre documento ed elemento radice. Il metodo **`End`** chiude tutto. Il metodo **`StartPulse`** apre l'elemento **`<Pulse>`**. Il metodo **`EndPulse`** gestisce l'elemento **`<WorkQueueState>`** e chiude l'elemento **`<Pulse>`** (la classe **`WorkQueue`** è definita nel file "**`WorkQueue.cs`**" che deve essere aggiunto al progetto). I quattro *handler* gestiscono i corrispondenti elementi XML (ad es., **`WorkStartedHandler`** gestisce l'elemento **`<WorkStarted>`**).

Nel **`Main`**, creare un **`Logger`** e un **`WorkQueue`** e agganciare i due eventi generabili dall'istanza di **`WorkQueue`** ai corrispondenti *handler* dell'istanza di **`Logger`**. Quindi, prima del ciclo di generazione degli impulsi invocare il metodo **`Start`** con argomento "**`../../Output.xml`**" e dopo il ciclo di generazione degli impulsi invocare il metodo **`End`**. Inoltre, nel corpo del ciclo, come prima istruzione invocare il metodo **`StartPulse`** e come ultima istruzione invocare il metodo **`EndPulse`**. Infine, nel metodo **`eventsGenerator_WorkArrival`** aggiungere alla **`workQueue`** (precedentemente creata nel **`Main`**) il **`work`** passato come argomento.

Per testare il codice, lanciare l'applicazione: il file di output deve contenere tutti gli elementi **`<Pulse>`**, **`<Enqueued>`** e **`<WorkQueueState>`** (che, a sua volta, deve contenere l'elenco dei *work* in coda). Si noti che, per ora, i *work* aggiunti alla coda non vengono mai rimossi.



Passo 3 – Definire la classe **`Worker`** (come da schema UML). La classe **`Worker`** descrive un "lavoratore" mediante un identificatore univoco (**`_id`**) e un eventuale *work* in corso (**`_work`**). In fase di creazione di una nuova istanza, si utilizzi il *field* statico **`_nextId`** per inizializzare correttamente l'identificatore univoco. Le proprietà **`Id`** e **`Work`** permettono di accedere ai corrispondenti *field*, mentre la proprietà **`IsWorking`** permette di sapere se il *worker* è occupato. I metodi **`OnWorkStarted`**, **`OnWorkDone`** e **`OnIdle`** devono scatenare i corrispondenti eventi. Il codice dei metodi **`Assign`** e **`Pulse`** è contenuto nel file "**`Worker.txt`**".

Nel **`Main`**, creare e inserire in una lista tanti *worker* quanti indicati nel file XML di *input*; per ogni *worker* agganciare gli eventi **`WorkStarted`** e **`WorkDone`** ai corrispondenti *handler* del *logger* e l'evento **`Idle`** al metodo statico **`Program.worker_Idle`** il cui codice è contenuto nel file "**`Worker.txt`**". Infine, nel ciclo di generazione degli impulsi, dopo l'invocazione del metodo **`Pulse`** sull'istanza della classe **`EventsGenerator`**, invocare il metodo **`Pulse`** su tutti i *worker* esistenti.

Per testare il codice, lanciare l'applicazione: il contenuto del file di output deve coincidere con quello del file "**`OutputAtteso.xml`**".