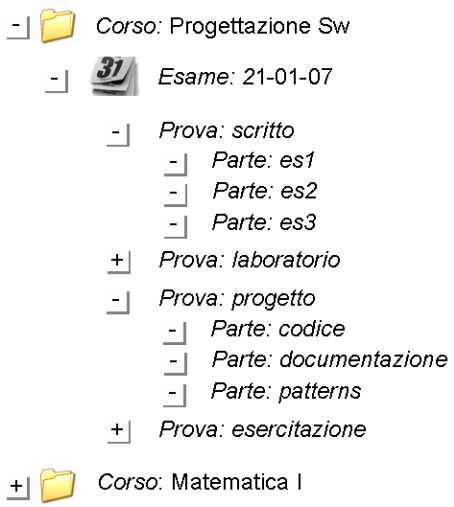
GradeBook

Progetto d'esame

Validità: giugno 2007 – dicembre 2007

Descrizione del problema



Obiettivo del progetto è la realizzazione di uno sistema distribuito per la gestione dei voti degli **esami** sostenuti dagli studenti universitari. **Docenti** e **Studenti** interagiscono con il sistema attraverso opportune interfacce grafiche rispettivamente al fine di inserire e visualizzare informazioni relative agli esami.

Ogni **esame**, prima di essere superato, impone il superamento di diverse **prove** quali ad esempio: prove scritte, progetti, tesine, ricerche, esercitazioni, esami orale, etc...

Il voto finale di un **esame** per uno **studente** S_k, deve poter essere calcolato come combinazione lineare dell'esito di ogni singola **prova** P_i (la più recente in ordine temporale):

$$voto(S_k,E_1)=w_1*voto(P_1)+..+w_n*voto(P_n)$$

Il voto attribuito ad ogni singola **prova sostenuta** P_i può dipendere dall'analisi di diverse **parti** p_{ij} della prova *Pi* sostenuta:

$$voto(S_k,P_i)=w_1*voto(p_{i1})+..+w_m*voto(p_{im})$$

Diventa quindi necessario avere la possibilità di associare una valutazione ed eventuali commenti ad ogni singola **parte** di una prova.

Ogni **docente** deve poter creare e configurare uno o più **corsi** e stabilisce per ogni **esame** quali sono le **prove** da sostenere. Un **esame** o **prova sostenuta** sostituisce gli eventuali esami o prova sostenute in precedenza.

Un **docente** può modificare solo i suoi esami o quelli in cui fa parte del gruppo di docenti e attribuire una valutazione alle prove sostenute, mentre può visionare e analizzare tutte le informazioni degli altri corsi.

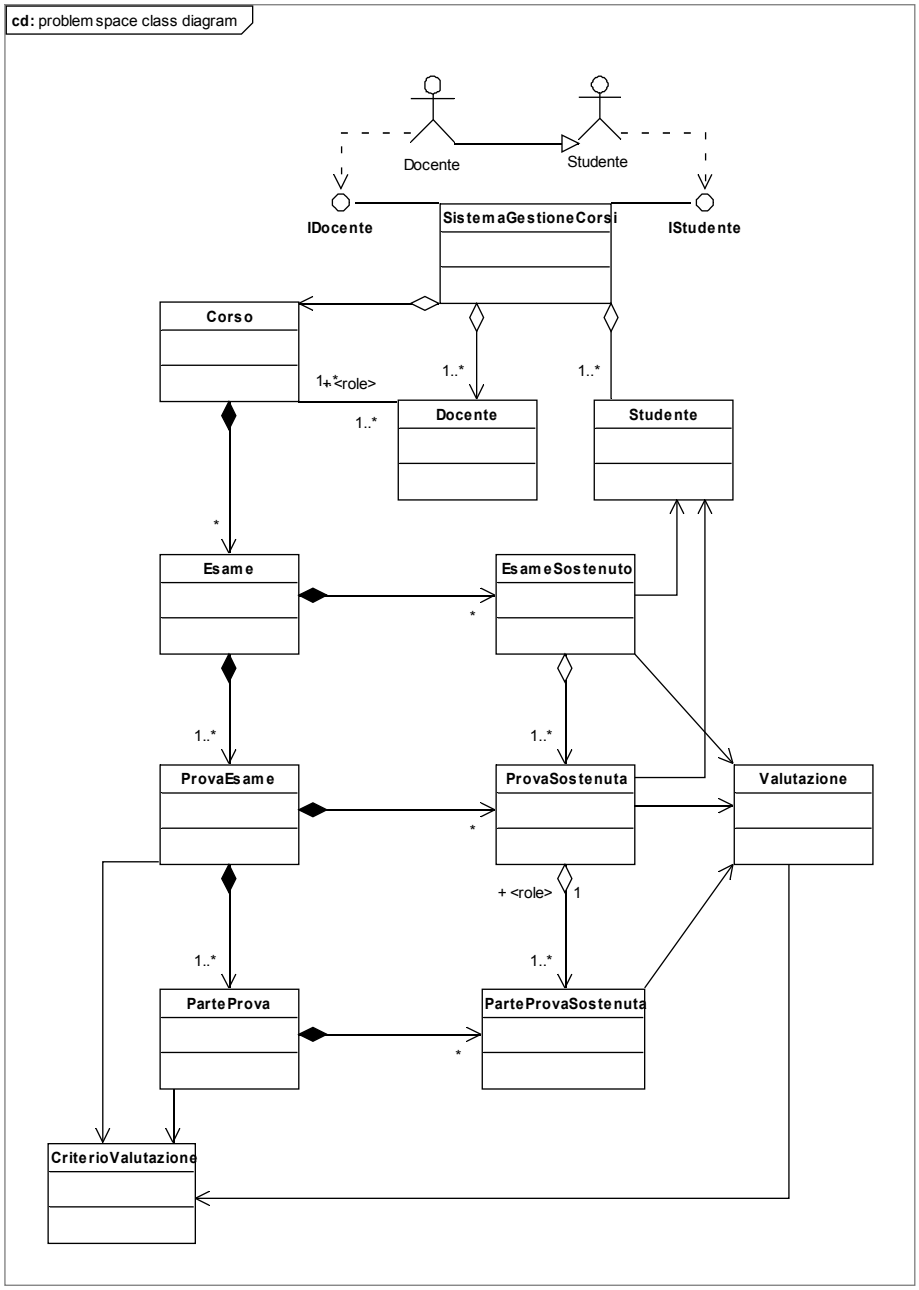
Uno studente può soltanto accedere in lettura a tutte le informazioni relative a corsi, esami e prove contenute nel sistema. In particolare deve poter

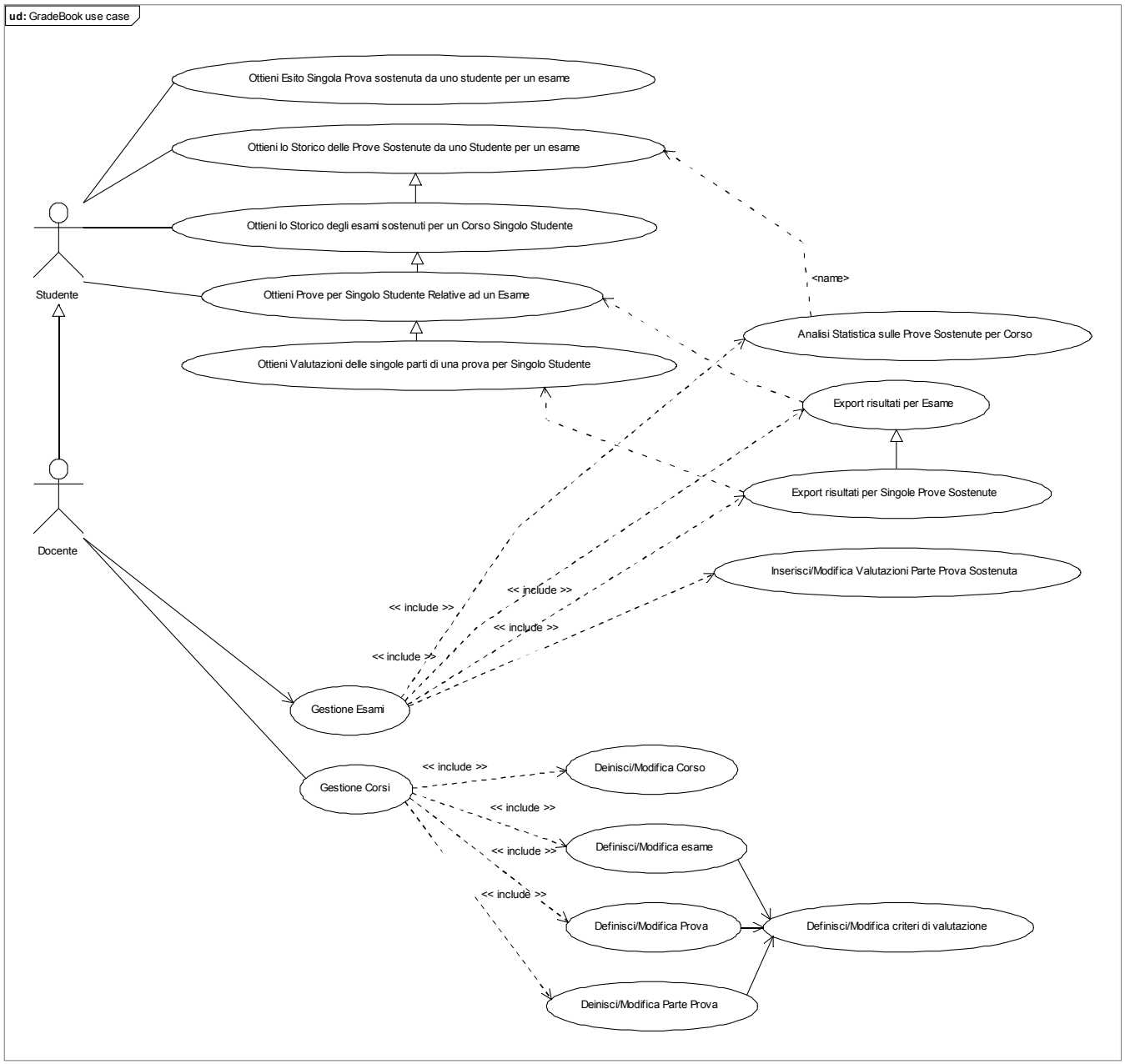
- analizzare lo storico (di uno studente) di una prova d'esame;
- analizzare lo storico (di uno studente) relativo ad un esame;
- deve poter calcolare la percentuale di studenti che hanno superato un esame in un particolare periodo di tempo
- deve poter calcolare delle statistiche sul singolo esame o su tutti gli esami...
- deve poter calcolare delle statistiche sul singolo studente (voto min e max) o su un gruppo di studenti .

Tutte le Informazioni devono poter essere esportate in HTML o in Txt e deve essere possibile aggiungere nuovi formati di export.
L'insieme degli studenti deve poter essere importato da un formato CSV (comma-separated values http://en.wikipedia.org/wiki/Comma_Separated_Values) dove i campi sono separati da ',', ma deve essere possibile aggiungere nuovi formati di import.
Il sistema deve poter salvare continuamente il suo stato sul disco in un particolare formato (scegliere tra serializzazione, files di testo, file XML, DB) ma deve essere aperto e dare la possibilità, a chi voglia implementare un nuovo formato, di cambiare la modalità di salvataggio. Il salvataggio serve per garantire un meccanismo di ripristino dello stato del sistema in caso di malfunzionamento, spegnimento del server, interruzione della corrente, etc. Diventa quindi necessario associare al salvataggio anche una funzione di ripristino dello stato del sistema.

Spazio del Problema

I diagrammi che seguono sintetizzano il problema sopra esposto.





NB: nello spazio della soluzione questi diagrammi potrebbero essere completamente diversi.

Cosa bisogna fare

Il progetto d'esame è uno strumento che consente di dimostrare la conoscenza e la padronanza di tecniche di progettazione e implementazione.

Il progetto viene quindi valutato in due fasi distinte e sequenziali (prima di iniziare la seconda fase bisogna aver superato la prima):

1. Il gruppo si iscrive all'appello per la presentazione del progetto (nello spazio della soluzione). Ogni individuo del gruppo presenterà la parte che intende realizzare. Qui verranno valutati:
    1. l'architettura del sistema (client-server)
    2. design pattern (consiglio: usare il pattern **composite** per rappresentare un generico EsameSostenuto e il pattern **strategy** per i vari formati di import, export e salvataggio dello stato del sistema)
    3. aspetti statici e dinamici del sistema (vedi esempio di documentazione UML)
    4. prototipi delle interfacce grafiche associati ai vari use-case.
2. Il gruppo implementa la soluzione e consegna (via email) il software per dimostrare di aver seguito le specifiche del progetto. Ogni studente facente parte del gruppo deve compilare la tabella che documenta la quantità di lavoro svolta (vedi esempio in appendice). Qui verranno valutati:
    5. soddisfacimento dei requisiti e documentazione UML (vedi esempio allegato)
    6. checkstyle
    7. design pattern implementati con UML
    8. qualità del codice sorgente
    9. documentazione javaDoc
    10. tabella con i tempi di sviluppo

N.B. Il progetto dovrà essere presentato al docente. E' pertanto richiesto allo studente di saper argomentare in modo opportuno le scelte progettuali e implementative adottate. Durante la presentazione una serie di domande da parte del docente punteranno a verificare l'effettiva padronanza delle tecniche utilizzate.

Progettazione della soluzione

Le attività di analisi e design devono essere adeguatamente documentate facendo uso del linguaggio UML. L'attività di modellazione porta dallo spazio del problema, allo spazio della soluzione. E' necessario documentare i criteri di raffinamento e le scelte progettuali (ad es. l'utilizzo di design patterns) che portano alla definizione dello spazio della soluzione.
È espressamente vietato produrre modelli UML ottenuti attraverso reverse engineering (cioè ricavando il modello UML mediante uno strumento di analisi del codice sviluppato).

Implementazione

Codice sorgente

- Struttura del codice: ogni volta che si identificano due estratti di codice sufficientemente simili sia nel codice di implementazione che negli intenti, il codice deve essere reingegnerizzato, e il codice duplicato eliminato portando a fattor comune le funzionalità condivise. Si consiglia l'utilizzo di strimenti di refactoring forniti da ambienti di sviluppo come Eclipse e NetBeans.

- Entità etichettate in modo significativo: non utilizzare identificativi "indecifrabili" (i, j, k, foo, pippo, e simili). I nomi delle variabili locali, dei metodi, delle classi devono essere significativi e rispecchiare il loro intento.
- Regole di stile: per indentazione, nomi, parentesi, commenti, etc. lasciatevi guidare da

checkstyle. Vedi anche <http://java.sun.com/docs/codeconv>.

Documentazione

JavaDoc. La documentazione JavaDoc è obbligatoria per i metodi e gli attributi con visibilità public o protected di ogni classe con visibilità public. Facoltativa per le classi con visibilità package e per i metodi e gli attributi con visibilità package o private. JavaDoc deve spiegare come usare una determinata classe o interfaccia, non come questa è stata effettivamente implementata, se questo può essere tenuto nascosto (Information Hiding).
Commenti al codice. Prima di commentare del codice perché altrimenti incomprensibile, tentare le seguenti strade:

1. Se si sono usati nomi indecifrabili, sostituirli con nomi che spieghino l'intento dell'entità (variabile locale, parametro, attributo, metodo, classe, package...)
2. Se il corpo di un metodo è troppo grande, decomporre il metodo in sotto metodi (eventualmente a visibilità private)
3. Se i metodi o gli attributi di una classe sono troppo numerosi, decomporre la classe in sotto classi (eventualmente a visibilità package)
4. Se la classe svolge uno specifico ruolo di un pattern, specificare il ruolo, il pattern e gli altri partecipanti (quali altri classi e che ruolo hanno); non descrivere il funzionamento del pattern.
5. Se il codice rimane comunque di difficile comprensione, inserire commenti espliciti.

Documentazione di sistema con UML. Il sistema è commentato principalmente tramite UML. UML deve far vedere la struttura statica del sistema ad un livello di astrazione maggiore del codice sorgente (Class diagram). UML deve far vedere la struttura dinamica del sistema, catturando le principali interazioni fra classi attraverso Sequence Diagrams, Interaction Diagrams, Object Diagrams, State diagrams. NB: non è necessario usare tutti questi diagrammi, basta che l'insieme prodotto sia sufficiente a rappresentare chiaramente il comportamento del sistema (sia dal punto di vista strutturale che comportamentale). Ogni pattern implementato deve essere documentato (vedi esempio di documentazione UML).

Strumenti di sviluppo

Obbligatori

Make: ANT <http://ant.apache.org/>

UML: Poseidon <http://www.gentleware.com> oppure ArgoUML <http://argouml.tigris.org/>

Code analysis & quality: CheckStyle <http://checkstyle.sourceforge.net> (utilizzando *sun_checks.xml* come file di configurazione)

Facoltativi

CMS: CVS / Subversion
IDE: Eclipse, NetBeans
Test di modulo: JUnit

Consegna

Il progetto va consegnato in formato compresso **ZIP** via e-mail al seguente indirizzo: ignazio.gallo@uninsubria.it. Nel progetto **non bisogna inserire** files di tipo .class, .bat, .sys, .sh, etc.. che, per motivi di sicurezza, potrebbero portare all'eliminazione del file allegato alla email da parte del nostro server di posta elettronica.
I progetti devono essere corredati da:

1. documentazione di sistema, documenti UML (vedi esempio allegato)
2. codice sorgente
3. file di build (targets obbligatori: compile, javadoc, run)
4. eventuali librerie necessarie alla compilazione e/o all'esecuzione
5. file RaccoltaDatiProgetto.xls compilato
6. file di README con eventuali annotazioni sulla compilazione

Il progetto deve compilare (tramite il file di build per *ant*) correttamente, una volta espanso in una qualsiasi directory. I progetti con errori di compilazione e/o di esecuzione non verranno valutati.
Se il progetto fa uso di una versione specifica di Java (ad esempio la 5 o la 6), specificarlo nel file di README e, quando possibile, forzare un controllo anche nel file di build.
Se il progetto fa uso di particolari librerie, usate in modo non standard, specificarlo nel file di README.
Se la struttura delle directory del progetto non è intuitiva, specificarla e dettagliarla nel file di README.

Appendici

Esempio di utilizzo di checkstyle

Ipotizzando di installare la versione più recente di *chechstyle* nella directory *D:\local\checkstyle-3.5* ed il codice sorgente del progetto si trova tutto all'interno della directory *src* allora i passi da seguire sono i seguenti:

```
set CLASSPATH=%CLASSPATH%D:\local\checkstyle-3.5\checkstyle-all-3.5.jar
cd <DIR_PROGETTO>
%JAVA_HOME%\bin\java.exe com.puppycrawl.tools.checkstyle.Main -c sun_checks.xml -r src
```

oppure utilizzando ant possiamo inserire un target per checkstyle nel file build.xml:

```
<property
   name="checkstyle.base"
   location="specify_location" /> <!-- specify location -->
<property
   name="checkstyle.jar"
   location="${checkstyle.base}/checkstyle-all-4.3.jar" />
<property
   name="checkstyle.configuration"
   location="${checkstyle.base}/sun_checks.xml" />
<taskdef
   resource="checkstyletask.properties"
   classpath="${checkstyle.jar}"/>
<property
   name="src"
   location="specify_src_location" /> <!-- specify source code location -->
<!-- checkstyle -->
<target description="CheckStyle sun_checks" name="checkstyle">
   <checkstyle config="${checkstyle.configuration}">
   <fileset dir="${src}" includes="**/*.java"/>
   <formatter type="plain"/>
   <formatter type="xml" toFile="specify_checkstyle_errors_file"/> <!-- specify
checkstyle errors file -->
   </checkstyle>
</target>
```

Il progetto consegnato non deve produrre alcun messaggio di warning con l'esecuzione di checkstyle.

Esempio di utilizzo di JavaDoc

Il modo più semplice consiste nell'inserire un nuovo target all'interno del file build.xml di ant:

```
<!-- javadoc -->
<target depends="init" description="Javadoc for my Project." name="javadoc">
   <mkdir dir="${javadoc.dir}"/>
   <javadoc destdir="${javadoc.dir}" packagenames="it.*">
     <sourcepath>
      <pathelement location="${src.dir}"/>
     </sourcepath>
   </javadoc>
</target>
```

ed eseguire il comando

<ANT_INSTALL_DIR>\bin\ant.bat javadoc

In alternativa, se i file sorgenti del vostro progetto si trovano nella cartella *src*, potete eseguire il seguente comando:

```
%JAVA_HOME%\bin\javadoc -d html -sourcepath src -verbose
```

I file html generati da javadoc si troveranno nella cartella html.

Esempio di descrizione della conduzione del lavoro

Processo di sviluppo

Il processo di sviluppo che si è seguito o a cui ci si è ispirati... Quando possibile dare una breve descrizione del processo.

Suddivisione progetto

Le macro attività in cui è stato diviso il progetto. Se le macro attività non sono autoesplicative inserire una breve descrizione della macroattività.
Esempio: Analisi sistema, Architettura e design del sistema (inclusa la documentazione UML), Implementazione riconoscimenti parametri di shell (inclusa la verifica), Implementazione shell interattiva (inclusa la verifica), Implementazione modulo di confronto (inclusa la verifica), Implementazione modulo di sincronizzazione (inclusa la verifica), Verifica dell'intero sistema, Altro, ...

Suddivisione lavoro

Compilare una tabella in cui, per ogni macro attività, per ogni partecipante al progetto, si indichi il tempo che è stato dedicato allo svolgimento. Se un gruppo vuole far rimanere la suddivisione del lavoro anonima, è possibile compilare la tabella inserendo al posto dei nomi degli identificatori fittizi (mr. pink, mr. white, mr. brown, ...).
Esempio:

	pippo	***pluto***	***tizio***
Analisi	1.45		
Modellazione e design		4	1
Implementazione modulo X		0,5	3
Testing modulo X	3		
...	...	...	...
Documentazione			
Altre attività			
...			

Esempio file di build di ant

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all" name="SushiBoatRestaurant">

<!-- init -->
<target name="init">
<property location="classes" name="."/>
<property location="." name="."/>
<property location="doc/api" name="javadoc.dir"/>
<property name="project.name" value="${ant.project.name}"/>
<property name="classes.dir" value="build" />
<property name="src.dir" value="src" />
<property location="${project.name}.jar" name="jar"/>
</target>

<!-- compile -->
<target depends="init" name="compile">
   <mkdir dir="${classes.dir}"/>
   <javac debug="true" deprecation="true" destdir="${classes.dir}" srcdir="${src.dir}"/>
</target>

<!-- jar -->
<target depends="init,compile" name="jar">
   <jar basedir="${classes.dir}" compress="true" jarfile="${jar}">
   <manifest>
      <attribute name="Main-Class" value="sushirestaurant.SushiBoatRestaurant"/>
   </manifest>
   </jar>
</target>

<!-- all -->
<target depends="init,jar" description="Build everything." name="all"/>

<!-- javadoc -->
<target depends="init" description="Javadoc for my API." name="javadoc">
   <mkdir dir="${javadoc.dir}"/>
   <javadoc destdir="${javadoc.dir}" packagenames="sushirestaurant.*">
     <sourcepath>
       <pathelement location="${src.dir}"/>
     </sourcepath>
   </javadoc>
</target>

<!-- run -->
<target depends="init,jar" description="Esecuzione del Ristorante Sushi" name="run" >
   <parallel>
   <java jar="SushiBoatRestaurant.jar" fork="true">
      <arg value="gm" />
      <arg value="2" />
   </java>
   <sleep seconds="5" />
   <java jar="SushiBoatRestaurant.jar" fork="true">
      <arg value="boat" />
      <arg value="10" />
   </java>
   </parallel>
</target>
</project>
```

Per eseguire questo file bisogna prima installare *ant* e poi eseguire uno dei seguenti comandi:

<ANT_INSTALL_DIR>\bin\ant.bat compile

<ANT_INSTALL_DIR>\bin\ant.bat run

<ANT_INSTALL_DIR>\bin\ant.bat jar