COMMENTO DEL DOCENTE AL TEMA D'ESAME DEL 12/07/2011

PREREQUISITI

Il corso di Fondamenti T2 dà per acquisita da Fondamenti T1, che ne costituisce il necessario prerequisito, la capacità di impostare algoritmi anche non banali, indipendentemente dal linguaggio di codifica e dalle specifiche strutture dati utilizzate.

PROLOGO AL TEMA D'ESAME

Come già nel compito precedente, a una persistenza piuttosto standard si accompagna un modello dei dati più complesso, ancorché la parte richiesta sia un'unica classe. La parte grafica è di media complessità, ma i componenti già forniti abbreviano i tempi di realizzazione del pannello meteo.

MODELLO DEI DATI

Sebbene il dominio dell'applicazione possa a prima vista apparire complesso, solo una classe (***GlobalForecast***) dev'essere realizzata: le altre sono fornite nello Start Kit. Vi sono due aspetti chiave: un costruttore privato con relativo metodo factory, che insieme incapsulano buona parte del problema; e l'algoritmo di media pesata. Completano il quadro alcune competenze collaterali, come la conversione stringa/numero/stringa necessaria per applicare i pesi alle diverse condizioni meteo, e – soprattutto – l'abilità di gestione delle date e dei calendari.

Il metodo factory riceve una collezione di previsioni puntuali relative a una singola città, ma riferite a più date; suo compito è quindi separarle, creando collezioni distinte per date distinte. A tal fine occorre prestare attenzione a confrontare *la sola componente giorno/mese/anno dell'entità* ***Date****, prescindendo dall'orario:* per questo il diagramma UML suggeriva la predisposizione di un metodo statico ***sameDate***, da definirsi sfruttando adeguati calendari [*non era accettabile l'uso di metodi deprecati della classe* ***Date****].* Fatto ciò, era banale costruire, tramite il costruttore privato, la previsione globale per ogni giornata, aggiungendola a un'opportuna struttura dati, da restituire come risultato finale.

Il costruttore privato riceve una collezione di previsioni puntuali relative a una singola città e a una ben specifica data: esse si differenziano quindi solo per l'orario entro la giornata. Il costruttore deve quindi scandirle una ad una, recuperando previsione e orario di ciascuna, per combinarle secondo l'algoritmo specificato. A tal fine è opportuno recuperare innanzitutto la prima previsione, il cui orario costituirà la base iniziale per il calcolo della media – *che, ricordiamolo, in generale non avviene su 24h, ma su un periodo minore, e precisamente fra tante ore quante ne intercorrono fra la prima previsione e la mezzanotte successiva* (ad esempio, se la prima previsione è alle ore 4.00, la media sarà su 20 ore; se la prima previsione è alle 11.00, la media sarà su sole 13 ore; e così via). Successivamente si dovrà iterare sull'insieme (ordinato per data), calcolando a ogni iterazione quante ore sono trascorse dalla previsione precedente e usando tale dato per calcolare il contributo alla media ponderata. Alla fine, il totale ottenuto andrà diviso per il numero di ore del periodo considerato, e – tramite una conversione inverso numero/stringa (facilmente effettuabile o con uno switch o, più opportunamente, con una mappa o un array di stringhe costanti adeguatamente predisposto), sfruttato per determinare la previsione globale richiesta.

Per quanto riguarda le strutture dati, era certamente comodo appoggiarsi a una mappa, avente per chiave la città, e per valore la lista delle corrispondenti previsioni; scelte diverse erano ovviamente possibili ma a prezzo di maggior onere nel gestirle.

Ciò conclude i primi 8/30 del compito.

PERSISTENZA

La gestione della persistenza era piuttosto standard: ***MyWeatherIconLoader*** doveva leggere un singolo oggetto (una mappa) dal file binario, per poi restituirlo [nota: in caso di problemi dovuti a differenti versioni del runtime Java, la cui classe javax.swing.ImageIcon è stata modificata nel versione 1.6.0_23 creando problemi in alcune installazioni del laboratorio, era comunque possibile bypassare il problema appoggiandosi al mock presente nel test, che caricava direttamente i file PNG in una mappa creata sul momento – un approccio meno rapido, ma version-independent]. Anche la parte di lettura da testo, gestita da ***MyForecastReader***, non era particolarmente complessa, dato che il formato del file era piuttosto standard - a linea singola e con pochi valori.

Ciò conclude la prima parte del compito, del valore di 16/30, realisticamente fattibile in 2h (2h30 al più).

GUI: CONTROLLER

Questa parte si appoggia in larga misura a quanto svolto in precedenza: in particolare, i metodi ***getCities*** e ***getIcon*** si basano sul loader e sulla mappa forniti come argomento al costruttore, mentre il metodo principale, ***searchForecast***, poteva utilmente avvalersi di due metodi ausiliari (non pubblici) ***searchGlobal*** e ***searchPuncual***, come suggerito dal

diagramma UML; al loro interno questi dovevano appoggiarsi ai servizi di ***GlobalForecast*** e precisamente a ***sameDate*** e ***calcGlobalForecast*** per far effettivamente svolgere le attività "intensive".

GUI: INTERFACCIA UTENTE

Anche questa parte si appoggia al pannello meteo fornito nello start kit. Inoltre, nonostante i componenti da aggiungere alla GUI siano diversi, si tratta di gestire sostanzialmente un'unica azione, ancorché scatenabile da due distinti tipi di eventi (evento di azione dai due radiobutton e dalla scelta città nella combobox, evento di cambio stato dal cambio data nel DateSpinner). Per questo poteva essere utile, come suggeriva il diagramma UML, fattorizzare in un unico metodo ***refreshForecast*** la logica di gestione dell'evento, rimappando su esso sia ***actionPerformed*** sia ***stateChanged***. Tale metodo doveva nell'ordine 1) recuperare la città selezionata dalla combobox, 2) recuperare la data dallo spinner, 3) ottenere dal controller le previsioni da mostrare (tramite ***Controller.searchForecast***, il cui terzo argomento è un boolean che stabilisce se si vuole la previsione globale o l'insieme di quelle puntuali, ed è facilmente ottenibile dallo stato attuale del radiobutton GLOBALE), 4) passare tale insieme di previsioni al ***WeatherPanel*** interno, tramite ***setForecast***, 5) chiamare il metodo di sistema ***validate*** per forzare l'aggiornamento dei vari pannelli.

(Stima tempi: non oltre 1 ora.) Ciò conclude il compito.

COMMENTO FINALE

Come sempre, il compito poteva apparire lungo, ma lo diveniva solo se si ci metteva a "scrivere codice a capofitto" senza prima riflettere costruirsi un'idea chiara delle relazioni fra le entità e delle funzionalità che ciascuna offriva. Conviene sempre dedicare almeno 20-30 minuti, all'inizio, a ragionare sul testo e a capire bene quali dati sono disponibili e dove, quali servizi sono offerti e da chi, come è sempre stato fatto durante tutte le esercitazioni. Può essere utile in tale fase farsi un piccolo riassunto e/o schema su carta, se ciò collima con le proprie abitudini; gli strumenti di sviluppo (Eclipse) potrebbero essere lasciati utilmente chiusi in questa fase, in quanto *non si guadagna tempo, ma anzi se ne perde, cominciando subito a scrivere codice senza riflettere prima in modo approfondito.*

Il compito non va preso come una "lotta contro il tempo": al contrario, occorre sforzarsi di mantenere la calma, *ricordando che il compito è costruito per essere risolto con intelligenza (ivi inclusi i suggerimenti impliciti nel diagramma UML..), non per creare difficoltà inutili*. Se una cosa appare (troppo) complessa, forse è perché la si sta guardando dal lato sbagliato.. e forse non si stanno sfruttando appieno funzionalità già esistenti e fornite.