**ESAME**

APPUNTI E COSE UTILI

IMPORT E EXPORT

* Import
  + Import->Existing projects from workspace->copia e incolla nome della cartella(più veloce)
  + NB importa cartella, non file zip
* Export
  + Export->Archive file->Scegli la cartella e il nome->Esporta (Se usi la libreria UniBs esporta anche quella)

LIBRERIA

* Build path->Projects->Add->UniBSLib
* NB non lo zip, il profe fa così quindi fallo anche tu
* NB **import** it.unibs.fp.mylib.\*; //Lo inserisce da solo il programma in realtà

JAVADOC

* /\*\* \* \* @param @returns \*\*/ (Vedi esempio tamaSbagliato)
* NB non farlo per i metodi privati
* Prima delle classi l’autore
* Project->Generate Javadoc

JUNIT

* Build path->Libraries->Classpath->Junit
* Src->new->JunitTest
* Nei test inizializza oggetti final(in carta alta però non lo fa) e fai semplicemente dei test
* I test chiamali come casi specifici (Cosa che sono) e esegui il metodo che vuoi controllare, non girarci intorno…
* Test, assertEquals/True/False(metodo da verificare,risultato aspetto)

ESERCIZI DEL PROFE:

-Punto-retta

- Cuori solitari

-Tamabase

-Polveri Sottili

-Archivio CD

-Tamazoo

-Carta più alta

-Lavatrici

-Titoli Azionari

-Conto Corrente

Simulazioni:  
-Morra cinese

-Massaia

-Azienda sanitaria

APPUNTI SULL’ESAME

* **NB LEGGI BENE LA CONSEGNA**
* commenta classi e metodi
* crea metodi static di utilità fuori dal main(?)
* Main snello ma non troppo(Menu e switch scelta di solito)
* NO stringhe che girano, fai costanti che abbiano senso
* NB Override toString, segna sopra @override (Utile quando in un print finale devi mettere tutti gli attributi etc), utile per stampare i dati di un oggetto, non fare 3000 print di qua e di la, ma richiama solo toString
* Costruttore con tutti gli attributi, poi fai un metodo in cui inserire i dati da passare al costruttore nella classe stessa ( NomeClasse creaNomeClasse(){…. return NomeClasse(); //Costruttore } )
* Fai classe Eccezione da invocare nel catch (Basta fare una classe che estende Exception e il costruttore che manda un messaggio attraverso super(messaggio) e poi richiamarlo tramite oggetto.getMessage())
* NB se non sai cosa può essere un oggetto da ereditarietà (Es. random pick) valuta bene come comporre il programma, magari cerca di utilizzare gli stessi metodi per tutte le classi, nel caso overloaded o overridden(Vedi Tamazoo). Se invece hai sotto controllo le sottoclassi (scelta arbitraria), procedi come in planetarium etc differenziando ogni sottoclasse a tuo piacere e tenendo i metodi in comune nella superclasse
* SALVATAGGIO FILE
  + Per salvare testo vedi “morra” (Vedi ProvaFile)
  + Per salvare oggetti vedi “TitoliAzionari” (Usa classe ServizioFile)
* Per l’organizzazzione del progetto se è a varie classi che interagiscono prendi spunto da AziendaSanitaria
* Se sono calcoli matematici/giochi, vedi morra cinese o quadrato magico, solitamente bastano poche classi… (Vedi anche in relazione al tempo che hai a disposizione)
* Se può servire…printStackTrace ti dice dove è/sono avvenuti gli errori (o quantomeno se dovesse chiedertelo lo sai almeno)
* NB Se un’informazione è implicitamente univoca, fai in modo che non venga ripetuta

Appunti da punto-retta

-quanti decimali-> %.n nel printf

-Override del metodo String toString() nella classe che deve usare il metodo

-String format funziona come un printf ma restituisce una stringa

-Può essere apprezzato il for abbreviato

Appunti cuori solitari

-costruttore che passa tutti gli attributi

-metodo static che prende in input tutti gli attributi in variabili e li posiziona negli argomenti del costruttore (Meglio nella classe stessa, limita i static…)

-variabile da inizializzare successivamente, metti null all’inizio (Fai un controllo Exception NullPointer?)

-Vedi la classe enum

-interazione tra due oggetti puoi farla nella classe stessa dell’oggetto (Come avevo fatto con la battaglia golem, forse meglio fare una nuova classe, però valuta…)

TAMABASE

-Costruttore con tutti gli attributi, poi fai un metodo in cui inserire i dati da passare al costruttore

-Esempio javadoc

-main dato solo dal menu e dalla creazione dell’oggetto

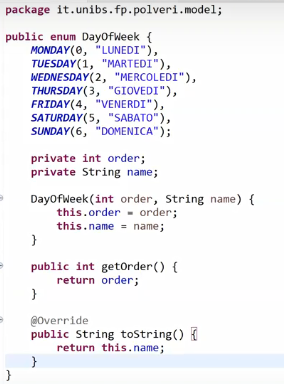
- Nei return boolean utile fare subito un return <condizioni> (non if(<condizioni>) return true else return false)

-se un valore potrebbe andare sotto zero, utilizza Math.max(0, numero);

-Guarda test

POLVERI SOTTILI (non hai il file, nel caso c’è il video su elearning, mi segno solo gli appunti)

-Classe enum per i giorni



-Per print con variabili fai un printf e metti a costante la stringa formattata (Con i %f, %s, …)

-Classe a parte Misura con il valore e il giorno

-Eccezioni (Lui usa throw, per comodità e perché sei abituato utilizza try catch)

-classe gestione menu->main snello

-NB leggi bene consegna e capisci bene cosa vogliono (hai fatto il programma in modo un po’ diverso…, ad esempio devi inserire manualmente ogni volta anno e numero della settimana)

-Nell’output su console sii più completo possibile (Es. se ci sono i giorni, scrivi il nome del giorno anche se non strettamente necessario…)

-Limita enum alle informazioni essenziali dell’oggetto

-Un vettore di valori=un vettore di giorni->Fai una classe misura…

-In un array è da inizializzare col costruttore ogni elemento(?) (Usa ArrayList che fai prima…)

ARCHIVIO CD

-NB il contenitore di CD richiamalo in una classe, non fare un array nel main

-Per trovare un elemento cercalo, se non lo trovi salvalo null e dopo verifica se è null oppure se effettivamente è stato trovato l’elemento

-Utile per organizzazione e gestione di classi

-Test, assertEquals/True/False(metodo da verificare,risultato aspetto)

TAMAZOO

-Ereditarietà->richiama il costruttore super( )

-Se hai vari tipi di sottoclassi per differenziarli magari inserisci un attributo “tipo”

-Override/Overload di tutti i metodi che li differenziano in modo che qualsiasi tipo sia e qualsiasi metodo richiami, esso sia presente

-usa protected su metodi/costanti che utilizzano anche le sottoclassi

-meglio istanziare direttamente nel costruttore

-metodi e variabile tamazoo static esterni al main (meglio fare una classe a parte Utility e dichiarare il tamazoo non static magari…)

-Utile per ereditarità e test

CARTA PIU ALTA

-se random fai uno switch e in base al numero che esce richiama un costruttore diverso

-per gli enum usa metodi che ritornino Classe.datoenum (ES. EsitoScommessa.VINTA), se non ti ricordi bene come usarli guarda

-Classe.values() puoi usarlo mettendo in un’altra classe un vettore Classe[ ] vettore, e poi come argomento attuale Classe.values() (Classe =nome della classe enum)

-Abstract class->usala se la superclasse non deve mai essere istanziata

- Se la superclasse non ha neanche un metodo in comune (in quanto a implementazione), tanto vale fare un’interfaccia… (Utile per enum, ad esempio vedi ValoreCarta)

-Per dichiarare un enum esso deve rispettare il costruttore (NB vedi esempio TipoMazzo)

LAVATRICI (Interfaces)

-Vedi per le interfacce se ti può servire, per il resto abbastanza inutile

TITOLI AZIONARI

-Per verificare se è la prima volta che apri il programma o no, verifica se il file esiste (file.exists() ,è boolean). (Ovviamente se è la prima volta che apri il programma ma c’è un file già esistente con lo stesso nome fai le tue valutazioni opportune…)

-File .dat utile per salvare Oggetti

-Per salvare Oggetti vedi il main, utilizza ServizioFile, che può essere utile al posto di salvare ogni volta una marea di stringhe o int…

-Fai una classe apposta il cui oggetto venga salvato e caricato dal file(Es Container)

-Utile il file in generale per vedere come funzionano salvataggio e scrittura su file

-NB classe **implements** Serializable

CONTO CORRENTE

-Classi eccezione personali

-Throw (Ma lancia eccezioni, piuttosto usa try-catch…)

-Controlla e verifica tutti i casi in un programma, es. se puoi prelevare, quanto puoi versare etc, se ci son possibili errori o fatti la classe Exception personale o usa un’eccezione già esistente…

-Salvataggio oggetti

TRAFFICO

-Classe astratta con metodi già definiti, se devi definirli per ognuno scrivi abstract, senza l’implementazione. (Vedi classe Elemento)

-Utile per rappresentazioni grafiche su console (classe Strada)

-NB argomento superclasse può essere sostituito da sottoclassi

MORRA CINESE (“morra” nei progetti)

-Nel main crea un oggetto per richiamarlo (te all’esame per semplicità nel main metti Menu e switch di scelta)

-Classe Mossa è una classe ibrida, mhh… Io piuttosto farei la enum Mossa con i suoi metodi e una gestioneMosse con i metodi static, altrimenti fai tutti i metodi static che fai prima… (Ad esempio comparaMossa poteva immettere come argomenti due mosse e rendere il metodo static)

-NB per estrarre un enum a caso

Mossa.*values*()[NumeriCasuali.*estraiIntero*(0, Mossa.*values*().length-1)];

In generale utile per avere un enum dato un “numero scelta”

- Salvataggio di testo su file, vedi come fanno. Più semplicemente potresti fare PrintWriter e File

AZIENDA SANITARIA

-NB controlla, se un codice è univoco, che non esista già (In generale se un’informazione è univoca, guardaci che sia effettivamente univoca)

-NB nelle scelte dello switch usa i metodi static del main, magari crea (l’ho già fatto nel mio progetto) una classe ManagerAziendaSanitaria che faccia la stessa cosa (però aggiungi ogni singola opzione, compresi gli InputDati etc, che non son da mettere nelle classi oggetto come hai fatto…)

-Nelle classi fai metodi corti e chiari, quelli più corposi con input, system out falli tutti nei metodi static

-Se devi fare un progetto con switch etc dacci un occhio perché è fatto bene, soprattutto per come è ordinato e organizzato

-NBF per i print dei dati di un medico o paziente fai il ToString

-Non si fanno tanti problemi a ripetere due volte un ArrayList, molto più comodo per la gestione di tutti i pazienti… (Però attento, sono oggetti diversi…)

-Evita di restituire indici coi metodi se si tratta di ArrayList, cerca di passare direttamente oggetti

-Genera tutti i get e set, anche se probabilmente manco li userai

-NB cosa che non sapevi, puoi fare remove(Oggetto) e non per forza con l’indice,è più comodo e veloce direi… (Funziona anche con gli oggetti, stai attento che deve essere lo stesso altrimenti non va…, puoi fare ogg1=ogg2 e se fai remove(ogg1) o remove(ogg2) sarà la stessa cosa)

Ad esempio il profe ha fatto una nuova List=List dei pazienti totali dell’ospedale. Ha fatto selezionare il paziente stampandoli tutti con vicino un numero e avendo così l’indice. Ha ricavato il paziente che gli serviva e l’ha fatto rimuovere al medico.

In alternativa avrebbe potuto fare immettere il numero di tessera sanitaria, fare trovaPaziente(), restituire il paziente e da lì rimuoverlo (ipotesi mia) ) //Se non sei sicuro che funzioni evita e usa gli indici dell’arrayList (piuttosto che non vada meglio farlo un po’ più lungo…)

-Per un maggiore ordine usa i for abbreviati(Oggetto x: vettore)

-Fai attenzione ai casi specifici e possibili Exception…

-NOTE PERSONALI pensavo di fare un ArrayList<Paziente> anche in AziendaSanitaria ma pensavo che fosse sbagliato "ripetere" gli oggetti, invece il profe l'ha fatto tranquillamente

Forse la soluzione migliore sarebbe stata fare l'ArrayList in AziendaSanitaria e a ogni medico passare un vettore di int contenenti i numeri di tessera sanitaria dei pazienti(Così da evitare di duplicare gli oggetti, cosa che mi hanno sconsigliato all'Arnaldo)

MEDAGLIERE OLIMPICO

-Progetto non completo ma ordinato e pulito nel complesso, se devi vedere per l’ordine dacci un occhio…

-Utile per vedere le classi Elenco

-Commenta le graffe

QUADRATO MAGICO

-Se hai una sola scelta fai yes or no, non un menu

-Se ci sono calcoli matematici fai poche classi normalmente…

METODI UTILI NELLA LIBRERIA UNIBS

CLASSE INPUTDATI

- *leggiUpperChar(Messaggio, elenco caratteri consentiti) //Verifica maiuscole e minuscole, non fa ignoreCase*

*-leggiIntero(Messaggio, min, max)*

*-leggiStringaNonVuota(Messaggio) //Abbastanza intuitivo che controllo faccia, verifica se funziona con spazio e tab*

*-YesOrNo*

ESTRAZIONI CASUALI

*-Abbastanza intuitivo… (solo con int)*

MYMENU

*-NB è una classe istanziabile, utilizza il costruttore mettendo il titolo e un vettore con tutte le alternative.*

*-Per evitare errore out of bounds magari inizializza il vettore direttamente es:*

*In alternativa voci puoi farlo anche come costante scrivendo*

*private static final String[] VOCI={…,…,…};*

String [] voci = {"scelta1", "scelta2"};

MyMenu menu=**new** MyMenu("PROVA", voci);

menu.stampaMenu();

*-è ordinato e fa risparmiare spazio nel codice, vedi su momento se utilizzarlo o no…*

*-scegli()->al posto di stampaMenu e InputDati*

METODI UTILI (Alcuni esempi fatti in ProvaCostruttore)

-StringBuffer

* Reverse(), stampa al contrario una stringa

-ArrayList

* Remove(Oggetto), puoi rimuovere un oggetto richiamandolo oppure uno con lo stesso riferimento.Insomma non inizializzare mai un nuovo oggetto per poi cercarlo, “Prendilo” da qualche parte… (Vedi ad esempio AziendaSanitaria del profe, dove Paziente p=elenco. Etc, quindi cercava/rimuoveva poi il paziente prendendolo dall’elenco)
  + ESEMPIO

ArrayList<StringBuffer> elenco=**new** ArrayList<StringBuffer>();

StringBuffer str1=**new** StringBuffer("ciao");

StringBuffer str2=**new** StringBuffer("ciao");

StringBuffer str3=str1;

elenco.add(str1);

elenco.remove(str2);

System.***out***.println(elenco.size());

elenco.remove(str3);

System.***out***.println(elenco.size());

//Stampa 1 e poi 0

NB se provi a rimuovere un oggetto non presente non da errore, anche se arrayList.size()==0

* IndexOf(Oggetto) NB vale il discorso del remove(Oggetto) Ritorna -1 se non trova l’oggetto
* Contains(Oggetto) come indexOf ma restituisce true/false