1. Cambiare il main nel build.gradle
2. Gradlew eclipse (capire cosa fanno questi comandi gradlew)
3. Gradlew build
4. Gradlew run

Organizzare sempre il progetto in package

Nel test plan no casi d’uso (quelli di patella) perché non li sappiamo fare ahah

Da qualche parte scrivere un commento con un’autovalutazione da mantenere (dandomi un voto all’americana? Con il perché della votazione)

Appunti:

1. Leggere i requisiti
2. Analisi dei requisiti (poche essenziali parole che mi vengono spontanee come analista)
3. Analisi del problema (poche essenziali parole che mi vengono spontanee come progettista)
   1. Come fanno a interagire le due componenti? -> quali tipologie di interazione ci sono?
4. Opzionale: dato il problema che componenti mi servono e come interagiscono
5. Architettura logica:
6. Piano di test: se ti mando fibo32 tu mi rispondi in una certa maniera -> si/no

INTERACTION

Tutte le entità che realizeranno questo concetto astratto di interaction dovranno obbedire a quel contratto, sono dei pojo? (è un acronimo play old(?) java object) Queste entità sono ancora oggetti e non attori

Sottolineare che IApplMessage segue uno standard interno ed è fortemente strutturato

Se volessi creare una richiesta non bloccante

* I produttori non sono vincolati ad essere scritti tutti nello stesso linguaggio
* Chi produce info è produttore, chi consuma info e consumatore, ma c’è un unico modo attraverso il quale fluisce l’informazione? Almeno due strade: io chiedo l’informazione (pull) oppure la do e basta. Quale è più ragionevole? Normalmente è il produttore a inviare le info
* Sono in dubbio? Chiedo al committente
* Evento: informazione emessa da un produttore senza avere conoscenza di chi la recepirà
* La comunicazione deve essere sincrona o asincrona. Quale è meglio? Asincrona. Che garanzia ho che venga ricevuta? Il consumatore deve darmi un feedback? Mi deve inviare un ack?
* Il consumatore è sincrono o asincrono, in modo fifo o no? Dipende perché se il produttore è fire and forget, a me non importa se è fifo o no

Io vedo solo il request e non il dispatch

Observer appiccicato al receiver (consumer) (non ne sono certissima)

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Tanti thread quanti sono i core

Dal contesto allo stato

Applicazione indipendente dalla natura dei dispositivi d’uscita. È il dispositivo che si adatta all’applicazione (aka le esigenze applicative), non il contrario (INVERSIONE DELLE DIPENDENZE)

Display dal punto di vista concettuale va discusso:

* Se è un componente architetturale (è un pezzo hardware)
* Ha una parte software da usare? Come le invio i messaggi? È un pojo o un actor\*? Di solito è un pojo e se è così avrò un’interfaccia e quindi avrò delle funzioni write/print/etc. Chi definisce l’interfaccia? Chi ha sviluppato il software nel display.
* \*domanda da farci sempre d’ora in poi
* Se è un actor devo sapere interazioni che fa (request, dispatch, event). Nel caso di display dispatch al 90% .

Per effettuare inversione delle dipendenze si utilizzano degli adapter. Per esempio nel caso della write di un display ci sarà un port adapter che mi permetterà di pilotare il display e non essere soggetta a quello che vuole il dispositivo

Al centro ci sono i requisiti

SistemaRilevamento

In blu sostantivi (entità) in rosso verbi (azioni)

Stazione, fotocamera, sonar, led : ENTITA’ 🡪 elenco inconsistente perché devo dire se sono pojo o attori

Stazione (attore perché deve impartire comandi (accendere/spegnere sonar)), fotocamera (pojo. A che attore è legato? Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Use cases (o history (in qualità di … farei questo ….): accendere il sonar (stazione) [verificare che si sia acceso]

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

Facade24 (?) guida per realizzazione parte output del progetto (la parte input varia sempre)

Ping = sonar . però il ping deve conoscere il pong per inviare il dispatch. Il sonar del sottomarino mica conosce … (l’ostacolo?)

Se qualcosa non parte controlla:

* Src (nelle proprietà del progetto)
* Plugin (devi copiare i plugin del prof nella cartella dropins di eclipse)
* Librerie (unibolibs)
* Main (nel build.gradle)

ServiceObserverCoap fa partire un server(?) che opera come observer

Dockerfile file da creare per creare l’immagine docker. Prima cosa da fare generare il file tar (per linux) gradlew distTar, poi il build e poi il run (stanno tutti nel dockerfile)

Core business del progetto sonar è farlo funzionare ovvero in questo caso scrivere i valori della distanza

Sonar usage mock per far vedere che è il cliente dall’esterno a inviare i comandi start/stop, ma l’unico ceh può veramente attivare e deattivare il sonar è il device

All’inizio voglio vedere le cose e me le modello io, una volta che ho capito tutto come funziona, sommergo delle parti

1. Creare il progetto (la cartella)
2. Dentro la cartella creata: gradle init e poi 1 2 invio invio
3. Apri dls, importa gradle project, nelle proprietà inserisci la natura java
4. Crea il package src
5. Dentro src crea il file nomeProgetto.qak
6. Dentro nomeProgetto.qak e scrivi almeno:
   1. Immagine che contiene testo, software, Carattere, Pagina Web

      Descrizione generata automaticamente
7. Modificare user docs con il mio nome e ci mettiamo copia e incolla dei requisiti e dell’analisi dei requisiti (se l’abbiamo già fatta)
8. Dopo aver scritto gli attori qak nel file .qak e aver messo il codice py, facciamo il runtime [# codice kotlin/py# ]

Lateinit serve per dire che verrà inizializzata più tardi

BoundaryWalk – Analisi dei requisiti

1. Formalizzazione della stanza per assicurarmi che sia un rettangolo
2. Posizione iniziale del virtualrobot
3. Sonar della stanza. Quando rilevano il robot, i sonar emettono le seguenti informazioni che sono delle stringhe json

Cril è il linguaggio che capisce il robot (concrete robot interaction language)

Messaggio di stato dice cosa sta facendo i robot (mi muovo in avanti, giro a destra etc), se sto fermo non invia messaggi di stato

Anche se non c’è scritto da nessuna parte, il passaggio davanti al sonar rappresenta un evento

Sostituita la parola Adapter con la parola Interpreter. Nasce così il linguaggio aril

PER APRIRE SCHERMATA ROBOTTINO:

1. Aprire docker
2. Startare il container
3. Fare localhost:8090

L’@override di update è indotto da chiunque estende ApplAbstractObserver

In questo caso lo step è lungo quanto la lunghezza del robot

Step sincrono si sblocca solo se lo step va a buon fine (la prossima mossa non è collision)

I file qaktt sono i file che non vuole eseguire al momento, se ti serve li rinomini in qak (e magari aggiungi uno spazio da qualche parte, salvi e rifai gradlew eclipse per rigenerare i file)

Per cambiare mappa del robottino:

1. Apri docker desktop, fai partire il container e connettiti a localhost:8090
2. Salvarsi l’id del container (il primo di solito) con il comando docker ps -a (nella cartella C:\Users\valen\OneDrive\Desktop\issLab24\**it.unibo.virtualRobot2023\node\WEnv\WebGLScene**)
3. Cambiare la scena con il comando docker cp sceneConfig.js CONTAINERID:/home/node/WEnv/WebGLScene/sceneConfig.js (sceneConfig.js è il file scene che vuoi)
4. Refresha la pagina del localhost e voilà
5. Per salvare lo stato docker commit CONTAINERID
6. Se poi nel progetto fai gradlew build e gradlew run lo fai partire

Quando c’è un’interruzione non solo devo tornare allo STATO dov’ero prima, ma devo tronare anche alla giusta TRANSIZIONE che stavo facendo quando mi sono interrotto

2 gui per l’esame: una che dice cosa deve fare e ricevere una risposta(camionista) e una gui per gestire e vedere lo stato esterno (amministratore del sistema)

Concetto dietro sta roba: relazione tra gui e applicazione

Spring 0: requisiti

Spring 1 attività che metto in campo per sviluppare il primo prodotto perfettamente funzionante che posso dare al committente. Soddisfa un sottoinsieme ben motivato di requisiti non tutti.

Spring2 soddisferà un altro sottoinsieme dei requisiti (ampliando lo spring1)

Modello a cipolla integrata, si parte dal core e spring dopo spring si costruisce l’intero progetto, ha come vantaggio che oltre ad essere facile da montare è anche facile da smontare

Che cos’è il ddlrobot? Se per esempio il committente può rispondermi che per ddlrobot intende il virtualrobot, mettiamo nella doc il link alla pagina che mi descrive il virtualrobot

Il commitente ci dice che è uguale a basicrobot, noi dobbiamo mettere il link a basicrobot24.html

Adesso ci chiediamo: è un POJO o un ATTORE?

Il virtualrobot **usa** il basic robot perché come si può SPECIALIZZARE un attore?

Non posso scrivere una facade direttamente con i qak perché il qak è un linguaggio di modellazione e non di web application. Per ovviare a questa cosa basta inserire la Facade nel codice qak, ricordandoti di aggiornare il file build.gradle perché aggiungendo Facade si aggiorneranno le librerie presenti in build.gradle24

Differenza tra clean architecture e la Domain Specific Languages è che qualunque applicazione è fatta da una sostanza e da cose ciappinose. Al centro dei pensieri di un progettista ci deve essere il dominio applicativo. Nel dominio non ci sono le interfacce, le gui, i pulsanti, ma per renderlo fruibile devo aggiungere le applicazioni web (che portano dal dominio ai casi d’uso) [ontologie del dominio un tempo messe nel database relazionale e ottenevo un’applicazione monolitica. Si arriva poi alla ontologia dei servizi (al giorno d’oggi). Potremmo anche dire microservizi, micro per il principio di singola responsabilità]

Al centro di ogni clean architecture ci sarà il core business dell’applicazione (dominio) che vengono costruite utilizzando i casi d’uso. Dato uno stesso dominio ci posso mettere tante applicazioni quante mi pare

In riferimento alla facade presenti in basicrobot24 noi siamo già avviati verso la clean architecture, perché ho il mio core business. Non corro il rischio mortale che si scriva nel controller dei pezzi di logica applicativa

Nel controller non ci devono essere pezzi di logica applicativa

Al centro di un’applicazione spring c’è il controller

Gestione dei messaggi nel controller: quando nella facade clicchiamo il bottone basicrobotip facciamo una post perché stiamo CAMBIANDO il contenuto. La get la facciamo quando vogliamo sapere qualcosa (tipo robotpos o quando facciamo localhost:8090>INVIO).

La logica applicativa della GUI (in questo caso del robot) non deve stare basicrobot [bah?]

R(right) è un dispatch mentre p(step) è una request

Il controller quando viene premuto un pulsante, deve trasformare la dispatch ricevuta utilizzando il linguaggio ARI (?)

Approccio agile:

Daily scrum : ogni giorno prima di iniziare a lavorare si fanno 15 minuti di pianificazione

Sprint review: il team die se ha raggiunto o no l’obiettivo

Sprint retrospective: il team si interroga sul processo di produzione stesso. Abbiamo organizzato bene lo sprint oppure potevamo fare di meglio? Per esempio è stato fatto l’userdoc? Il prof si rifiuta di leggere uno userdocs in cui c’è solo linguaggio naturale (rischio di incomprensioni)

Un modello è qualcosa che mette a fuoco alcune parti e ne tralascia altre (a differenza del programma)

Un modello comincia a nascere nella nostra testa quindi poi dobbiamo parlare in un LINGUAGGIO comune per comunicarlo. Deve essere chiara la differenza tra il CONCETTO e la sua RAPPRESENTAZIONE. Modello nasce in virtù di semplificare la comunicazione non ambigua tra esseri umani

Ci sono un numero infinitamente espandibile (non infinto anche se significa la stessa roba) di rappresentazioni di uno stesso modello

Per spazio concettuale si può intendere il COME fare una cosa. Le dimensioni di un possibile spazio concettuale sono tre: STRUTTURA, INTERAZIONE e COMPORTAMENTO. Quale dei tre è più importante? Nessuno perché sono tre facce della stessa cosa equivalenti. Gli insiemi dei modelli che descrivono un sistema

Attraverso MOF posso descrivere dei nuovi metamodelli. Concettualmente il prof, da ingegnere del software, ha preso il METAMODELLO che descrive il linguaggio qak, che ha dato a noi e noi scriviamo dei MODELLI di qak

Plugin servono anche a introdurre nuovi concetti nel linguaggio(?) Nel senso che non è solo il modello(?) a descrivere i concetti del linguaggio (?), ma è anche l’applicazione che può definire nuovi concetti attraverso le cose nei plugin

TEMA FINALE 24

Alla fine dello spring0 il prof vuole avere:

* definizione di un possibile modello dei requisiti
* definizione di rilevanti piani di testing funzionale
* ripartizione dei requisiti in classi di importanza / priorità
* stima dei tempi di sviluppo/produzione.

Tutto questo in forma comprensibile al prof e alla macchina.

Tre parti: struttura, interazione e comportamento. I requisiti vanno analizzati sotto questi tre punti di vista

Entità: software system service, OpRobot, Incenerator, service area, wasteStorage, AshStorage, RollPackets, MonitoringDevice, BURNIN, BURNOUT, Wastenin, ashout, led, sonar, messaggi come ActivationCommand, scaleInfo, sonarInfo

Ora mi interrogo e sviscero ogni entità

L’inceneritore è un’entità attiva perché riceve e invia messaggi. Ora mi chiedo sono dispatch, request o event? Da requisito le informazioni emesse dall’inceneritore (il fischio) sono un evento (fischia e lo sentono tutti); l’input da requisito non lo posso dire, perché so che gli arriverà un messaggio, ma non so né cosa mi arriva né da chi mi arriva, anche sul scaleInfo/sonarInfo non posso dire niente. Posso dire qualcosa sulla sua responsabilità: lui comincia a incenerire quando percepisce che c’è qualcosa in ingresso e finisce dopo un tempo BTIME. Adesso posso fare un modello eseguibile dell’inceneritore

Core business del sistema è incenerire, quindi devo mettere su un sistema per far vedere come funziona e in una prima istanza lo posso far vedere con cose simulate aggiungendo mock

Tutte le volte che vengono proposti dei nuovi requisiti si fa un nuovo sprint. Tipo se per simulare imposto un valore fisso della bilancia e dopo però passo ad analizzare questa problematica, faccio un altro sprint

Dobbiamo decidere se è l’oprobot a ricevere le cose oppure il sistema (wis). Il prof all’oprobot darebbe poche responsabilità. La responsabilità del business ce l’ha tutta il wis