RELAZIONE PROGETTO “Qontainer”

Georgiana Uglea 1121248

ISTRUZIONI COMPILAZIONE

La compilazione del progetto richiede un file project (.pro) diverso da quello ottenibile tramite l’invocazione del comando “qmake -project”.
Quindi, e’ opportuno utilizzare il file contenuto nella cartella del progetto chiamato “Qontainer.pro” affinche’ il progetto possa essere compilato ed eseguito adeguatamente. Inoltre nella cartella *build,* creata dal compilatore durante la compilazione del progetto, è necessario copiare il file “Container.xml”, locato nella principale del progetto: esso contiene infatti i primi *device,* sui quali e’ possibile iniziare subito ad effettuare operazioni.

AMBIENTE DI SVILUPPO

Sistema Operativo: Ubuntu 16.04 / Windows 10
IDE: QT 5.9.0/ 4.7.0
Compilatore C++: GCC 8.1.0
Standard di Compilazione: C++11

INTRODUZIONE

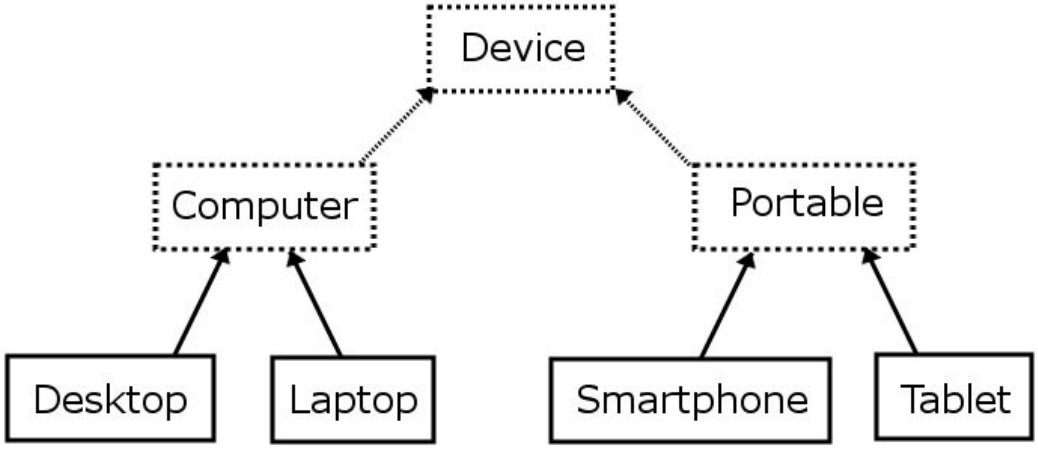
Il progetto in questione è un magazzino di *devices*, che possono essere di tipo:

- Desktop
- Laptop
- Smartphone
- Tablet

Il programma permette di inserire, modificare, cancellare e ricercare devices. Ogni device contiene le sue informazioni principali, salvate in un apposito file che verrà caricato all’inizio dell’esecuzione e che verra’ aggiornato in base all’operazione fatta.

DESCRIZIONE DELLA GERARCHIA UTILIZZATA

La gerarchia utilizzata in questo progetto è caratterizzata da una struttura lineare a tre livelli con derivazione pubblica.
La classe base astratta è “Device” da cui derivano le due classi astratte “Portable” e “Computer”. Le classi concrete “Smartphone” e “Tablet” implementano la classe astratta “Portable”; tale comportamento e’ speculare anche per le classi “Desktop” e “Laptop” che implementano la classe virtuale pura “Computer”.



Device

I tipi di dato utilizzati da *Qontainer* derivano tutti indirettamente da una classe chiamata “*Device”*. Essa, oltre al costruttore, implementa alcune funzioni base di utilità generale per tutti i tipi derivati, cioe’ i metodi di get e di set (tranne il metodo *get* di cui parleremo dopo). *Device,* inoltre, e’ caratterizzata da attributi comuni a tutti i tipi dispositivi elettronici trattati in questa gerarchia, come ad esempio la marca ed il modello del dispositivo, la quantita’ di RAM, e cosi’ via. La classe si definisce virtuale pura in quanto non implementa tutti i suoi metodi: alcuni di questi sono definizioni di metodi virtuali necessari per il corretto comportamento della gerarchia. Questo per garantire che, in fase di run-time, sia chiamato il giusto metodo della giusta classe.

Il metodo *get* funge da metodo ”*polimorfo*”, cioè il parametro della funzione è una stringa contenente il nome del campo da prelevare; viene successivamente restituita una stringa contenente il valore richiesto. Questo metodo è risultato molto utile nell’implementazione della GUI che ad esempio nella ricerca, contiene una QString con il nome del campo da ricercare.
Il metodo *details* restituisce semplicemente una stringa che contiene tutte le informazioni dell’oggetto, correttamente indentate, per evitare un enorme carico dell’interfaccia grafica nel mostrare ogni campo dell’oggetto.

SOTTOCLASSI ASTRATTE “Computer” e “Portable”
La sottoclasse “Computer” aggiunge un ulteriore campo chiamato “Tipologia”, che è tipico solo delle sue classi derivate “Desktop” e “Laptop”.
Anche la sottoclasse “Portable” contiene campi tipici e utili solo alle sue classi derivate “Smartphone” e “Tablet”, come ad esempio la capacita’ della batteria, la dimensione del display, la fotocamera, ...
La due classi principalmente servono a differenziare due macro categorie di device e rendono più chiara e funzionale la gerarchia.

SOTTOCLASSE CONCRETA “Desktop”
Questa classe è derivata pubblicamente da “Computer” e possiede tre campi propri:

- wattMin, contiene il minimo numero di watt che necessita l’alimentatore per funzionare correttamente.
- nHard, intero che specifica quanti hard disk contiene il computer
- uso, stringa che specifica l’uso del Desktop in questione, ad esempio “Domestico”, ”Lavorativo”, ecc…

Questa classe, essendo concreta, implementa tutte le funzioni virtuali delle sue superclassi astratte.

SOTTOCLASSE CONCRETA “Laptop”
Comportamentalmente speculare a Desktop, *Laptop* è derivata pubblicamente da “Computer” e possiede dei campi propri:

- batteria, double contenente il numero di mAh della batteria dello stesso laptop
- pollici, contiene il numero di pollici dello schermo del portatile

- peso, misura in kg il peso totale del dispositivo.

Anche questa classe implementa i metodi virtuali ereditati delle sue superclassi.

SOTTOCLASSE CONCRETA "Smartphone"

La sottoclasse è derivata pubblicamente dalla classe virtuale pura "Portable" e possiede un solo campo proprio: **connettività**, una stringa che specifica la massima generazione di rete a cui il cellulare riesce a connettersi, ad esempio 3G,4G,ecc…

Ovviamente, implementa tutti i metodi virtuali delle sue superclassi.

SOTTOCLASSE CONCRETA "Tablet"

La sottoclasse Tablet è derivata pubblicamente da " Portable" e possiede come campo proprio un booleano che specifica se il dispositivo è usufruibile con una scheda SIM oppure no.

Implementa tutti i metodi virtuali delle sue superclassi.

CLASSE TEMPLETIZZATA "Container"

Container è un template di classe che rappresenta una struttura simile ad una *std::list*, cioè contiene una classe interna privata "Nodo" che ha come campi:

- info: un campo di tipo generico "T" che contiene l'informazione del nodo stesso
- next: puntatore di tipo nodo ad un nodo successivo.

Inoltre, nella parte privata di *Container* troviamo il suo unico campo proprio *first*: un puntatore al primo nodo del container.

Nella parte pubblica della classe, oltre ai consueti costruttore e distruttore, e' presente:

- un metodo *size*, che restituisce il numero di nodi del container di invocazione;
- un metodo *empty*, restituisce true se e solo se il contenitore su cui è invocato è vuoto;
- un metodo *find(T x)*, restituisce true se nel container esiste l'oggetto x.

Inoltre, sono presenti due template di funzione:

- *find(U x)*, un metodo che sulla lista di invocazione di tipo Container<T>, ricerca solo ed esclusivamente gli oggetti di sottotipo U>T, quindi restituisce un Container<U> contenente tutti gli oggetti di tipo "U";
- *erase(U x)*, un metodo che cancella dalla lista di invocazione tutti e soli

gli oggetti che sono “castabili” a U.(Ad esempio da una lista di Device cancella tutti i Tablet).

Infine, la classe Container possiede una classe Iteratore che serve a per poter utilizzare gli oggetti di tipo Container al di fuori della classe stessa. La classe Iteratore ridefinisce i seguenti operatori : incremento prefisso, uguaglianza, dereferenziazione, accesso, disuguaglianza. Contiene anche i soliti metodi *begin()* ed *end()*.

E’ stato deciso di non implementare un’ulteriore classe Iteratore costante in quanto non sarebbe stata utilizzata: gli iteratori sono esclusivamente di utilita’ dei metodi che portano modifiche al container. Quindi, considerato cio’, non e’ stato implementato alcun iteratore costante.

CODICE POLIMORFO

- Nel file “Container.xml” il metodo *gatherDevice(QXMLStreamReader)* ha come tipo di ritorno “Device” ma restituisce un tipo “Specializzato”, in base alla lettura del tag.
- Nel file “Container.xml” il metodo *saveDevices(Container<Device*>*)*, riconosce a run-time tramite un dynamic_cast il tipo specifico del device.
- Nella view “*InserisciTablet*”,”*InserisciSmartphone*”,”*InsertDesktop*” e ”*InsertLaptop*”,utilizzano il metodo XMLUtility::saveDevices() con parametro di tipo sottoclasse.
- In “*modifyelement.cpp”*, attraverso l’iteratore degli oggetti della lista, viene castato dinamicamente il puntatore per conoscere a run-time il tipo dell’oggetto e visualizzare i campi corretti.

FORMATO DEL FILE PER IL SALVATAGGIO/CARICAMENTO

Il formato che è stato impiegato per la realizzazione del progetto è XML.

Qt mette a disposizione le classi QXMLStreamReader e QXMLStreamWriter per leggere e scrivere su file di estensione .xml, ma non è stato sufficiente utilizzarle poiché non permettono ad esempio la modifica del valore interno ad un tag.

Quindi ho utilizzato anche la classe QDomDocument e le sue sottoclassi in modo da poter ricercare e modificare nel file alcuni valori.

Nello specifico, il file “*XMLUtility*” contiene le seguenti funzioni:

- *gatherDevice(QXmlStreamReader& x),* funzione necessaria per leggere le informazioni del tag *device* e restituisce un oggetto di tipo device.
- *loadDevices(),* funzione che dato il file "*Container.xml*" restituisce la lista di dispositivi descritta nel file stesso.
- *saveDevices(Container<Device*>* x),* funzione che salva nel file .xml tutti gli oggetti del container x.
- *ModifyElement(int pos, std::string tag, std::string valore),* funzione che data la posizione,il nome del campo e il valore del campo dell'oggetto, sostituisce il valore vecchio con il valore nuovo.
- *DeleteElement(std::string tipo x),* funzione che cancella dal file .xml tutti e soli gli oggetti di tipo.(Ad esempio cancella solo i Laptop)
- *DeleteElementField(std::string tipo,std::string campo, std::string valore),*funzione che dato il tipo,il campo ed il valore, cancella tutti gli oggetti con quegli attributi.(Ad esempio cancella tutti i Tablet con memoria=16)

Il progetto dispone di un file di default, caricato in automatico all'esecuzione del programma e modificato in base alle operazioni svolte durante l'utilizzo.

TEMPO IMPIEGATO

Il tempo necessario per la realizzazione del progetto presentato è difficile da conteggiare esattamente, ma si aggira intorno alle 50 ore complessive, dovute principalmente all'apprendimento del linguaggio XML integrato alla comprensione della classe QT QDomDocument per la modifica del file Container.xml.
Suddivisione:
Analisi preliminare del problema 5 ore,
Progettazione modello e GUI 10 ore,
Apprendimento libreria Qt 10 ore,
Codifica modello e GUI 15 ore,
Debugging 5 ore,
Testing 5 ore.