# Logic Testing

## File AccessoTest

In tale file si è testato il funzionamento delle funzioni di login e registrazioni della classe “PC-Ready”.

Prima di eseguire i test descritti in seguito, viene eseguita una funzione di setup che genera delle mappe di clienti ed amministratori contenenti degli utenti di prova necessari per il controllo dei test, inoltre viene anche recuperata l’unica istanza del sistema PC-Ready presente.

**Login Test:** In questo test è stato visionato il corretto funzionamento della funzione effettuaLogin() del sistema PC-Ready, la quale prende in ingresso l’email dell’utente, la password e soprattutto la tipologia con il quale l’utente specifica se sta effettuando l’accesso come cliente o come amministratore. Dopo aver specificato chi sta effettuando l’accesso, tramite uno switch-case, effettuerà una distinzione in due casi. Nel primo caso ad effettuare l’accesso è un amministratore, in questo caso quest’ultimo verrà ricercato all’interno della mappa degli amministratori memorizzata dal sistema. Nel caso dell’utente fa la stessa operazione, ma la ricerca avverrà nella mappa dei clienti memorizzata. Per effettuare il testing in esame, le mappe clienti ed amministratori sono state generate in maniera provvisoria, e passate al sistema, il quale tramite la funzione effettuaLogin(), precedentemente indicata, ha controllato la presenza di un utente, anch’esso generato per effettuare il test, all’interno della mappa. Il controllo del funzionamento della funzione è stato effettuato con il comando, fornito da JUNIT5, assertNotEquals(), con il quale sono stati paragonati il messaggio restituito dalla funzione effettuaLogin() e il messaggio di default che la stessa funzione restituisce nel caso in cui il login vada a termine con esito negativo.

**Registrazione Test:** In questo test è stato visionato il corretto funzionamento della funzione richiediRegistrazione() del sistema PC-Ready, la quale prende in ingresso nome, cognome e l’email dell’utente oltre che alla sua password e al conferma\_Password necessario per assicurarsene il corretto inserimento. Dopo aver specificato tali informazioni, la funzione genererà un nuovo cliente con questi attributi e lo inserirà all’interno della mappa dei clienti memorizzata dal sistema. Per effettuare il testing in esame viene per prima cosa generato un nuovo cliente ed inserito nella mappa dei clienti tramite la funzione specificata in precedenza, e si controlla che l’inserimento è andato a buon fine attraverso il comando, fornito da JUNIT5, assertNotNull(), con il quale controlliamo che, all’interno della mappa è presente l’utente appena inserito. Poiché la funzione richiediRegistrazione() impedisce l’inserimento di un nuovo utente con un email già presente nella mappa, viene effettuato un controllo tramite il comando, fornito da JUNIT5, assertEquals(), con il quale controlliamo che in risposta alla nuova richiesta di inserimento è stato restituito il messaggio di errore di default.

## File AcquistoHandlerTest

In tale file si è testato il funzionamento della classe “AcquistoHandler”, la quale gestisce tutte le operazioni dei casi d’uso UC5, ovvero si occupa di tutte le operazioni di creazione di un nuovo carrello e di un ordine che permetterà poi di concludere l’acquisto di un componente o di una configurazione/bundle.

Prima di eseguire i test descritti in seguito, viene eseguita una funzione di setup che genera un cliente, generato appositamente per il test, necessario per potergli poi associare un carrello ed un ordine.

**Aggiungi al carrello Test:** In questo test è stato visionato il corretto funzionamento della funzione aggiungiInCarrello() dell’AcquistoHandler, la quale ha il compito di inserire una nuova componente o configurazione all’interno del carrello di un utente. Tale funzione può avere due sviluppi diversi, distinte tramite uno switch-case. Nel primo caso, in cui viene inserita nel carrello una nuova configurazione, essa viene aggiunta insieme alle copie dei singoli componenti che la costituiscono, controllandone la disponibilità. Nel secondo caso, che si ha con l’aggiunta di una singola componente, il procedimento è lo stesso del primo sono che il controllo e l’aggiunta della copia viene fatta per un singolo componente. L’obiettivo del testing è proprio quello di controllare che tali funzioni di inserimento nel carrello e controllo del numero di copie funzionino correttamente. Tale controllo viene effettuato generando un nuovo carrello e inserendo in esso tre copie di uno stesso componente di cui però ne sono disponibili solo due nel catalogo. Tramite il comando, fornito da JUNIT5, assertTrue(), viene controllato proprio che il numero di componenti inserite nel carrello è di due controllando quindi che sia l’inserimento che il controllo sul numero delle copie sia andato a buon fine.

**Aggiungi configurazione Test:** Questo test ci permette di controllare il corretto funzionamento dell’inserimento nel carrello di una configurazione, inoltre tale inserimento viene accompagnato da un ulteriore doppio inserimento di una componente “CPU” per controllare, anche in questo testing, il corretto funzionamento della gestione dei doppioni. La funzione dell’AcquistoHandler interessata in questo test è sempre aggiungiInCarrello() che già abbiamo analizzato nella descrizione precedente. Il testing si basa sulla generazione di una configurazione partendo dalla generazione delle sue componenti, fatto ciò la configurazione viene inserita all’interno del carrello insieme alle due componenti uguali di cui però si ha una sola copia disponibile. Per controllare che tutto è andato a buon fine, tramite il comando, fornito da JUNIT5, assertTrue() viene controllato che le componenti inserite nel carrello sono effettivamente due, la configurazione e una sola CPU.

**Ordine Test:** In questo test è stato visionato il corretto funzionamento della funzione terminaAcquisto() dell’AcquistoHandler, la quale si occupa di generare, prendendo in ingresso indirizzo, città e CAP della spedizione, un nuovo ordine, per le componenti inserite nel carrello di un cliente corrente. per effettuare il testing di tale funzione sono stati passati in ingresso dei dati casuali per terminare l’ordine e successivamente tramite il comando, fornito da JUNIT5, assertEquals(), è stato controllato che l’ordine è stato effettivamente generato paragonando la citta dell’ordine creato con la città che è stata scelta durante la creazione di quest’ultimo. Poiché i risultati coincidevano abbiamo avuto la conferma della creazione di un ordine destinato a quella città.

**Ottieni ordine Test:** In questo test è stato visionato il corretto funzionamento della funzione ottieniOrdineCliente() dell’AcquistoHandler, la quale si occupadi elencare tutti gli ordini di un cliente avendo accesso alla sua lista degli ordini data la sua email. Tale funzione si basa sulla funzione getListaOrdiniCliente() che si occupa di ottenere, sempre tramite la mail, la lista completa degli ordini. Nel testing, tramite il comando, fornito da JUNIT5, assertEquals(), viene paragonata proprio la lista ordini di un cliente, di cui si ha piena conoscenza, con la lista ottenuta proprio con la funzione ottieniOrdineCliente(). Poiche il comando assertEquals() ci da esito positivo, ne deduciamo che le due liste sono uguali.

**Aggiorna ordine Test:** In questo test è stato visionato il corretto funzionamento della funzione aggiornaInformazioni() dell’AcquistoHandler, la quale si occupadi modificare le informazioni su di un suo ordine prendendo in ingresso i dati da modificare e l’id dell’ordine in cui devono essere cambiati. Viene sfruttata la funzione del sistema PCReady modificaOrdine() che permette proprio di cambiare le informazioni di un ordine. Per effettuare il test abbiamo usato il comando, fornito da JUNIT5, assertNotEquals(), per paragonare un ordine di un determinato cliente e lo stesso ordine dopo aver effettuato le modifiche dei suoi dati proprio tramite la funzione presa in esame. Poiché il comando assertNotEquals() da esito positivo, ne deduciamo che l’ordine è stato modificato correttamente.

## File CompatibilityCheckerTest

In tale file si è testato il funzionamento della classe CompatibilityChecker, la quale ha il compito di verificare la compatibilità tra le varie componenti inserite nella configurazione.

In quanto le logiche di controllo compatibilità, nonostante le diversità tra le componenti, sono molto similari tra loro, si è deciso di testare solo i controlli di maggior impatto nell’assemblaggio di una configurazione o bundle.

Dunque, ciascuna delle funzioni sotto riportate, testa uno specifico override della funzione “controlloComponente(Componente comp)”: tali override prendono in ingresso un componente specializzato e la lista delle componenti attuali della configurazione, ai fini di poter eseguire un controllo con tutte le componenti attualmente presenti della configurazione.

**CPU Incompatibile Test:** tale test prevede, una volta inizializzato l’handler Configurazioni nel setup, l’inserimento nella configurazione di una Motherboard e di una CPU incompatibile con la suddetta motherboard, per poi provare l’inserimento di una CPU compatibile. Dunque, mediante una funzione “assertEquals()” si controlla se il secondo componente della lista della configurazione risulta essere proprio la CPU compatibile che si è provato ad inserire.

**Motherboard Incompatibile Test:** tale test, dopo aver rimosso la motherboard precedente, mantiene una CPU di socket “AM4” nella configurazione, e prova ad inserire una motherboard di socket “FCLGA1151”. Dunque si re-inserisce la motherboard con socket “AM4” e si controlla mediante la funzione “assertEquals()” se il secondo componente della lista della configurazione risulta essere proprio la Motherboard compatibile con socket “AM4” che si è provato ad inserire.

**Gpu Incompatibile Test:** tale test prevede un inserimento in una configurazione, che già contiene una motherboard e una CPU, di un Case con 4 slot disponibili e di due GPU: la prima occupa 2 slot case, mentre la seconda occupa 6 slot (ovviamente non esiste una tale GPU standard in mercato, si esagera solo per testare la funzione di controllo… ). Mediante una funzione “assertTrue()” si controlla, mediante un contatore, che la configurazione attuale contenga solo una GPU, ovvero solo la GPU che occupa 2 slot case.

## File ConfigurationHanlderTest

In tale file si è testato il funzionamento della classe “ConfigurationHandler”, la quale gestisce tutte le operazioni dei casi d’uso UC1 e UC2, ovvero si occupa di tutte le operazioni di creazione di una configurazione o di un bundle.

In quanto la creazione di una configurazione sfrutta le stesse funzioni della creazione di un Bundle (a meno della funzione “infoConfigurazione”), i primi 4 test riguardano la creazione di una configurazione.

Prima di eseguire i test descritti in seguito, viene eseguita una funzione di setup che inserisce dei componenti manualmente all’interno del catalogo utilizzato dal configuration handler.

**Seleziona Categoria Test:** tale test controlla il corretto funzionamento della funzione “selezionaCategoria”, e controlla mediante una funzione “assertNotNull” che la mappa restituita dalla funzione “selezionaCategoria” non sia nullo.

**Seleziona Componente Test:** tale test controlla il corretto funzionamento della funzione “selezionaComponente”, difatti si cerca di ottenere una RAM id pari a 3. Mediante una funzione “assertNotNull” si verifica che il componente restituito dalla funzione “selezionaComponente” non sia nullo.

**Conferma Componente Test:** tale test controlla il corretto funzionamento della funzione “confermaComponente”, cercando di inserire nella configurazione corrente la RAM precedentemente selezionata. Mediante una funzione “assertTrue” si controlla che il primo componente della lista di componenti della configurazione attuale sia effettivamente di categoria “RAM”.

**Conferma Termine Assemblaggio Test: :** tale test controlla il corretto funzionamento della funzione “terminaAssemblaggio” e della funzione “confermaConfigurazione”, dopo aver correttamente inserito un set di componenti compatibili nella lista componenti di una configurazione. Mediante una funzione “assertEquals” si controlla che la configurazione appena creata abbia una lista di componenti uguale alle componenti appena inserite.

**Crea Bundle Test:** tale test controlla il corretto funzionamento della funzione “terminaAssemblaggio”, della funzione “infoConfigurazione” e della funzione “confermaConfigurazione”, nel caso di creazione e di inserimento di un bundle nel catalogo. Mediante una funzione “assertTrue” si controlla che il numero dei Bundle contenuti sia effettivamente pari ad 1, ovvero che sia stato effettivamente inserito il bundle appena creato.

**Doppia Motherboard Test:** tale test controlla che la funzione “terminaAssemblaggio” impedisca effettivamente l’inserimento di un componente tale da essere unico nella configurazione, provando ad inserire una seconda motherboard nella configurazione. Nonostante tale operazione sia di responsabilità della classe “compatibilityChecker” si vuole effettivamente controllare se vi sono errori dovuti alla gestione del risultato dei controlli della funzione “terminaAssemblaggio”. Mediante una funzione “assertTrue” che prende in ingresso un contatore, si controlla che la configurazione che si sta creando contenga effettivamente 1 e una sola motherboard.

**Componente Incompatibile Test:** tale test controlla che la funzione “terminaAssemblaggio” impedisca effettivamente l’inserimento di un PSU tale da non fornire abbasta potenza per i restanti componenti selezionati per la configuazione. Nonostante tale operazione sia di responsabilità della classe “compatibilityChecker” si vuole effettivamente controllare se vi sono errori dovuti alla gestione del risultato dei controlli della funzione “terminaAssemblaggio”, cosí come si è visto per il precedente test. Per tale motivo si elimina il PSU precedente e lo si sostituisce con un PSU che fornisce solo 50W, e lo si inserisce in configurazione. Mediante una funzione “assertFalse” si controlla che la funzione “terminaAssemblaggio” ritorni false, dovuto al fatto che il PSU non è compatibile con i componenti scelti.

## File GestisciComponentiHandlerTest

In tale file si è testato il funzionamento della classe GestisciComponentiHandler, la quale ha il compito gestire la creazione, la selezione e la rimozione di una determinata componente. Si occupa anche della creazione di nuove copie di una specifica componente e della generazione di promozioni da applicare.

**Seleziona Componente Test:** In questo test è stato visionato il corretto funzionamento della funzione selezionaComponente() di GestisciComponenteHandler, la quale si occupa semplicemente di estrarre una componente dal catalogo dato il suo id. Nel Test controlliamo proprio che, dato un componente che siamo certi essere presente all’interno del catalogo, questo venga trovato e restituito dalla funzione presa in esame. Tale controllo è stato effettuato tramite il comando, fornito da JUNIT5, assertNotNull(), che ci permette di controllare se è stata effettivamente trovata nel catalogo una componente corrispondente a quella cercata.

**Crea Componente Test:** In questo test è stato visionato il corretto funzionamento della funzione creaComponente() di GestisciComponenteHandler, che, prendendo in ingresso i dati necessari, dipendenti da che tipo di componente si vuole creare, si occupa di generare la componente richiesta e di aggiungerla al catalogo. Nel Test controlliamo proprio che il componente che creiamo noi appositamente nel test venga inserito correttamente nel catalogo. Per fare ciò sfruttiamo la funzione selezionaComponente() del handler, di cui abbiamo testato il funzionamento in precedenza, per controllare, tramite il comando, fornito da JUNIT5, assertNotNull(), che nel catalogo, dopo la creazione, è presente l’elemento appena generato.

**Crea Copie Test:** In questo test è stato visionato il corretto funzionamento della funzione creaCopia() di GestisciComponenteHandler, la quale si occupa di generare, dato un numero n in ingresso, una quantità pari a tale numero di copie di un determinato componente. Le copie vengono poi inserite all’interno di una lista che conterrà tutte le copie di quel componente. Nel Test controlliamo proprio che queste n copie vengano effettivamente generate. Per fare ciò sfruttiamo il comando, fornito da JUNIT5, assertNotEquals(), con il quale paragoniamo la dimensione della lista di componenti di un componente corrente prima e dopo l’inserimento. Poiché il test da esito positivo abbiamo stabilito che le dimensioni delle due liste sono diverse e che ciò è dovuto all’inserimento in essa delle nuove copie.

**Set Promozione Test:** In questo test è stato visionato il corretto funzionamento della funzione setPromozione() di GestisciComponenteHandler, la quale ha il compito di impostare una percentuale di sconto ad un determinato componente. Nel test controlliamo proprio che la promozione viene impostata correttamente e per fare ciò, preso un componente generato appositamente per il test, controlliamo, tramite il comando, fornito da JUNIT5, assertNotEquals(), che il valore del suo attributo promozione, memorizzato in due variabili prima e dopo aver utilizzato la funzione setPromozione(), risulti essere cambiato, facciamo ciò proprio paragonando le due variabili appena create.

**Rimuovi Componente Test:** In questo test è stato visionato il corretto funzionamento della funzione rimuoviComponente() di GestisciComponenteHandler, che ha il compito di rimuovere una componente dal catalogo dandogli in ingresso l’id del componente da rimuovere. Per effettuare tale test abbiamo sfruttato il comando, fornito da JUNIT5, assertNull(), con il quale abbiamo controllato che all’interno del catalogo, dopo la rimozione di una componente, di cui eravamo certi della sua presenza nel catalogo, tale componente non fosse più presente.