**Proiect Calitate si Testare Software**



Profesor Coordonator: Alin Zamfiroiu

Student: Madalin-Mihai Picu

Grupa 1065

1. **Definirea si justificarea pattern-urilor implementate:**
2. Singleton = design pattern ce permite crearea unei singure instante ale unui obiect. Acesta este folosit atunci cand intr-o aplicatie se poate folosi o singura instanta a unei clase( de exmplu o conexiune la baza de date ce se poate face o singura data si folosita in mai multe parti ale aplicatiei).
3. Observer = design pattern pentru exprimarea relatiilor dintre obiecte(relatie de dependent 1 la multi). Este folosit atunci cand un obiect isi schimba starea si influenteaza alte obiecte(obiecte dependente), acestea fiind notificate de modificarile suferite de obiect(subiect). Acest design ofera o mai buna orgaizare a comunicatiei dintre clase.
4. Builder = design pettern pentru crearea unor instante noi. Acesta este foarte util pentru ca ofera intr-o maniera foarte usoara si rapida schimbarea unui atribut al unui obiect sau creearea unui obiecte setand doar anumite atribute.(de exempu vreau sa creez un nou angajat setand doar 2 atribute din 5 posibile).
5. Chain of responsibility = design pattern pentru rezolvarea unei cerinte. Daca cerinta respective se duce pe un anumit obiect care nu stie sa o resolve, pt ca problema sa nu ramana nerezolvata, este trimisa mai departe catre alt obiect(ca un lant). Am folosit-o in proiect pentru a gestiona mai usor o sarcina de la un departament la altul.
6. Composite = design pattern folosit atunci cand vrem sa inglombam un grup de obiecte similare unui singur obiect. Acest design este unul structural (exemplu un director ce contine atat fisiere, cat si directoare ce contin la randul lor fisiere). In proiect l-am folosit pentru a pune intr-o cusca atat animale cat si custi de animale(de exmplu un papagal in colivie, langa o girafa)
7. Façade = design pattern ce camufleaza mult cod(tot codul care se doreste a fi executat, fiind mult, se pune intr-o functie dintr-o clasa si se apeleaza doar acea functie). In proiect l-am folosit pentru monitorizarea unei zile obisnuite din cadrul gradinii zoologice.
8. Simple factory = design pattern creational ce ajuta la construirea obiectelor. In proiect l-am folosit pentru a crea tipul angajatului (cu / fara experienta).
9. Strategy = design pattern ce stabileste la runtime executia unei functii. L-am folosit in proiect pentru a stabili modul in care angajatii sunt platiti.
10. **Definirea si detalierea metodelor testate prin Unit Testing**
11. Metoda dataLaCareSePlatescSalariatii() -> metoda ce stabileste daca s-a depasit sau nu data de efectuare a platii. Metoda am testat-o dandu-I ca parametru ziua 10(ziua in care se face plata).
12. Metoda penalitat()i -> metoda ce stabileste penalitatile pe care le incaseaza departamentul daca nu au facut platile la timp. Testez aceasta metoda aplicand o penalizare de 10 lei .
13. Metoda prima() -> metoda ce returneaza prima pe care trebuie sa o dea angajatilor. Aceasta se calculeaza ca fiind procent 3% din salariul de baza
14. Metoda noulSalariu() -> metoda ce calculeaza noul salariu pe baza primei calculate din metoda prima().
15. Metoda cheltuieliCuSalariile() -> metoda ce returneaza salariile totale pe care gradina zoological trebuie sa le plateasca angajatilor. Aceasta primeste ca parametru salariul de baza al fiecarui angajat.
16. Metoda verificaRaportAnimaleAngajati() -> metoda boolean-a ce returneaza true daca numarul de angajati(ingrijitori) este acelasi cu numarul de animale si false daca nu este. Aceasta metoda are scopul de a determina daca sunt mai multe animale decat angajati pentru a se asigura ca niciun animal nu ramane nesupravegheat.
17. Metoda calculeazaNrPui() -> metoda ce returneaza numarul de pui care se afla in gradina zoologica.
18. Metoda verificaVarsta() -> metoda ce returneaza true sau false daca varsta unui angajat este corecta.
19. Metoda medieVarsta - > metoda ce calculeaza media de varsta a angajatilor.
20. Metoda identificaPui -> metoda ce adauga intr-o lista puii animalelor.
21. **Definirea si descrierea Test Case-urilor:**

Test Case-urile reprezinta scenarii de test. Ele sunt clase in care se definesc diverse metode de test avand scopul de a testa functionalitatea unei metode. De exemplu vreau sa vad ca o metoda imi returneaza la un anumit input un anumi output. Daca metoda este scrisa correct, testul va fi ok, daca nu testul va da fail. Astfel imi pot da seama daca metoda respective este sau nu implementata correct.

Test case-ul ma ajuta sis a vad daca metoda respective a suferit modificari nedorite intre timp(exemplu acum o saptamana metoda mergea, a modificat o intre timp un coleg fara sa testeze foarte atent si acplicatia nu mai merge. Prin test case vad care a fost modificarea si o pot corecta).

1. **Definirea si descrierea Test Suite-ului**

Test Suit-ul reprezinta colectie de Test Case-uri. Test Case-ul executa doar metodele din el, in timp ce Test Suit-ul executa toate Test Case-urile , implicit toate metodele acestora.

Test Suit-ul este bun pentru a vedea toate metodele odata, fiind foarte rapid de identificat daca este o problema.

1. **Descrierea sumara a functionalitatilor aplicatiei cu referire la pattern-urile** **si metodele testate**

Alicatia reprezita un program de gestiune al unei gradini zoologice.

Gradina dispune de 3 departamente(curatenie, mancare si HR) fiecare avand atributiile sale. Angajatii pot fi de 2 tipuri: cu/fara experienta. Lucrul acesta este implementat prin design pattern-ul SimpleFactory.

De asemenea, gradina are si animale de diferite tipuri. Obiectele de acest tip sunt construile cu ajutorul Builder-ului.

Animalele sunt inchise in cusca. Acestea pot fi singure in cusca sau impreuna cu alte animale(in cusca sau nu). De exemplu intr-o cusca poate fi o girafa, in alta cusca pot fi 2 elefanti si in alta cusca poate fi un elefant si un papagal in colivie.

Acest lucru este reprezentat in aplicatie cu ajutorul pattern-ului Composite.

Angajatii au anumite atributii de facut: departamentul de curatenie sa faca curatenie, cel de mancare sa hraneasca animalele si HR se ocupa de administrativ.

HR –ul le face pe cele mai multe, acesta distribuind sarciniile angajatilor printr-un Strategy. Cu ajutorul pattern ului Chain of Responsibility angajatii sunt anuntati de la unul la altul de noul task.

De asemenea, HR-ul are obligatia de a anunta angajatii cand se efectueaza plata salariilor. Acest lucru este realizat prin Observer.

Pentru a vedea un raport al unei zile normale de functionare, folosesc un Façade ce face raportul cu activitatile din timpul zilei. Cao rice gradina are vizitatori iar acest lucru este monitorizat.

De asemenea, se percepe o taxa de intrare, biletul avand acelasi pret in orice circumstanta si de aceea este creat cu ajutorul Singleton-ului.

Hr-ul pe langa aceste atributii trebuie sa intocmeasca tot felul de evidente: de exemplu penalizari, prime, cheltuieli salariale, tipuri de angajati si tot ce tine de acestia.