CTS Grail

Calitate si testare Software (Academia de Studii Economice din Bucure□ti)

1. Care este sintagma cheie pentru utilizarea design pattern-ului Singleton?

-clasa abstracta

-instanta abstracta

-familie de obiecte

-instanta statica

-instanta unica

2. Care varianta de Singleton, dintre cele enumerate, este recomandata sa fie implementat?

-Static block initialization

-Enum

-Lazy initialization

-Eager initialization

-Thread Safe

3. Ce sunt Design Patterns?

-Imagini vectoriale asemanatoare

-Serii de probleme

-Sabloane de probleme

-Sabloane de proiectare

4. Design pattern-ul Factory presupune utilizarea:

-unei singure instante dintr-o clasa

-unei metode de copiere a obiectului intr-un alt obiect

-apelurilor in cascada pentru metodele de modificare ale valorilor atributelor

-unei interfete comune pentru toate clasele de obiecte dintr-o familie

5. Design pattern-ul Singleton presupune utilizarea:

-apelurilor in cascada pentru metodele de modificare ale valorilor atributelor

-unei metode de copiere a obiectului curent intr-un alt obiect

-unei singure instante dintr-o clasa

-unei interfete comune pentru toate clasele de obiecte dintr-o familie

6. Design pattern-urile creationale ajuta la:

-extinderea claselor abstracte

-initializarea atributelor statice

-implementarea interfetelor

-initializarea obiectelor

-implementarea functiilor statice

7. Este recomandat sa folosim Singleton atunci cand:

-Avem un singur atribut

-Avem o instanta unica a clasei

-Avem o familie de clase

-Avem o singura interfata si mai multe clase

-Avem o singura clasa

8. In cate categorii sunt impartite design pattern-urile orientate obiect conform GoF?

-5

-4

-2

-3

-1

9. Simple Factory este folosit pentru:

-familii de clase

-clase complexe

-obiecte complexe

-clase concrete

-clase abstracte

10. Singleton si Simple Factory sunt design pattern-uri:

-structurale

-creationale

-de stare

-comportamentale

11. Care dintre urmatoarele design pattern-uri sunt structurale?

-Adapter, Singleton, Facade

-Proxy,Builder,Decorator,Facade

-Adapter, Decorator, Facade

-Factory, Facade, Adapter

12. Care sunt participantii (sau elementele) design pattern-ului Adapter?

-Adapterul compus, adapterul frunza si adapterul concret

-Adapterul abstract, adapterul concret si fabrica

-Clasa utilizata, fabrica si adapter-ul

-Clasa utilizata, clasa noua si adapter-ul

13. Design pattern-ul Builder este folosit atunci cand:

-se construiesc obiecte complexe cu foarte multe atribute

-se construiesc obiecte unice

-se construiesc obiecte cu ajutorul unei metode statice

-se construiesc obiecte complexe cu foarte multe metode

-se construiesc obiecte asemanatoare

14. Design Pattern-ul Prototype face parte din categoria:

-Structurale

-Comportamentale

-Creationale

-De stare

15. Design pattern-ul Prototype presupune utilizarea

-unei singure instante dintr-o clasa

-unei interfete comune pentru toate clasele de obiecte dintr-o familie

-apelurilor in cascada pentru metodele de modificare ale valorilor atributelor

-unei metode de copiere a obiectului curent intr-un alt obiect

16. Design pattern-urile creationale ajuta la:

-initializarea obiectelor

-initializarea atributelor statice

-implementarea interfetelor

-implementarea functiilor statice

17. Din categoria de Design Pattern-uri Creationale fac parte urmatoarele:

-Builder, Factory, Prototype

-Singleton, Factory, Facade

-Singleton, Command, Builder

-Prototype, Template Method, Builder

-Template, Decorator, Prototype

18. Utilizarea design pattern-ului Facade presupune:

-realizarea unui modul de creare de obiecte complexe

-realizarea unui wrapper pentru adaptarea claselor existente

-realizarea unui wrapper pentru a ascunde complexitatea unei librarii

-realizarea unei metode de gestiune a tuturor obiectelor dintr-o familie

19. Care dintre urmatoarele design pattern-uri sunt structurale?

-Adapter, Singleton, Facade

-Adapter, Decorator, Facade

-Factory, Facade, Adapter

-Proxy, Builder, Decorator, Facade

20. Facade este util cand:

-Se doreste simplificarea unui proces

-Se doreste adaugarea de functionalitati

-Niciuna dintre variantele de mai sus

-Se doreste crearea de obiecte cu multe proprietati optionale

21. La ce se utilizeaza Facade?

-Pentru framework-urile open-source disponibile

-Pentru a elimina din functionalitatile framework-ului

-Pentru a adapta o clasa la o alta clasa

22. Ce tip de design pattern este Facade?

-Structural

-Comportamental

-Creational

23. Ce reprezinta numele Facade?

-Pentru a utiliza framework-urile, se poate folosi aceasta fatada, fara a fi necesara cunoasterea tuturor claselor, a metodelor si a atributelor din cadrul framework-ului

-Realizeaza o fatada pentru framework-urile foarte complexe

-Toate variantele

-Usureaza lucrul cu framework-uri foarte complexe

24. Care este deosebirea dintre Decorator si Adapter?

-Decorator adauga si functionalitati noi

-Adapter adauga si functionalitati noi

-Nu este nicio deosebire

-Adapter defineste metoda build()

25. Ce tip de design pattern este Decorator?

-Creational

-Structural

-Comportamental

26. Care sunt participantii dintr-un Adapter?

-Clasa existenta, adapterul

-Clasa existenta, clasa utilizata, adapterul

-Clasa existenta, clasa adaugata, adapterul

27. In ce situatii se foloseste Adapter?

-Ori de cate ori nu se doreste modificarea codului existent

-Ori de cate ori este necesara conlucrarea mai multor framework-uri

-Ambele variante

28. Cum este implementat Adapter-ul de obiecte?

-Clasa Adapter contine o instanta a clasei existente si implementeaza interfata la care trebuie sa faca adaptarea

-Clasa Adapter mosteneste clasa existenta si implementeaza interfata la care trebuie sa faca adaptarea

-Clasa Adapter mosteneste clasa existenta si implementeaza interfata la care trebuie sa faca adaptarea

-Clasa Adapter contine instante ale celor doua clase existente

29. Care sunt cele doua tipuri de Adapter?

-Adapter de clase si Adapter de obiecte

30. Care sunt participantii (sau elementele) design pattern-ului Adapter?

-Adapterul compus, adapterul frunza si adapterul concret

-Adapterul abstract, adapterul concret si fabrica

-Clasa utilizata, fabrica si adapter-ul

-Clasa utilizata, clasa noua si adapter-ul

31. Ce design pattern trebuie sa folosim pentru adaugarea de noi functionalitati unei clase existente deja, fara a modifica in codul existent

-Decorator

-Facade

-Builder

-Adapter

32. Din ce categorie fac parte design pattern-urile: Facade, Adapter si Decorator?

-Structurale

-Creationale

-Comportamentale

33. Utilizarea design pattern-ului Facade presupune:

-realizarea unui wrapper pentru a ascunde complexitatea unei librarii

-realizarea unui modul de creare de obiecte complexe

-realizarea unui wrapper pentru adaptarea claselor existente

-realizarea unei metode de gestiune a tuturor obiectelor dintr-o familie

34. Care dintre urmatoarele afirmatii nu este adevarata?

-Clasele NodFrunza nu implementeaza metodele de adaugare si stergere a unui nod

-Clasele composite contin o lista cu elemente de tip ComponentaAbstracta

-In lista de obiecte de tip ComponentaAbstracta se pot adauga si sterge noduri

-Composite este utilizat la crearea meniurilor aplicatiilor

35. Cand este folosit design pattern-ul Composite?

-Atunci cand doreste adaugarea de noi functionalitati claselor existente

-Atunci cand se doreste modificarea functionalitatii unui obiect la runtime

-Atunci cand este nevoie sa se utilizeze clase din doua framework-uri diferite

-Atunci cand este necesara crearea unei structuri ierarhice prin compunerea de obiecte

36. Ce tip de design pattern este Composite?

-Creational

-Structural

-Comportamental

37. Care este corelatia dintre Composite si Decorator?

-Ambele ascund, intr-un fel, clasa existenta

-Construirea de obiecte este gestionata de o clasa

-Nodurile Composite pot fi privite ca Noduri Frunza decorate

-Legaturile dintre clase

38. Composite este util atunci cand:

-Se doreste construirea unei singure instante

-Se doreste construirea de obiecte cu multe proprietati ce nu sunt obligatorii

-Se doreste crearea unei structuri ierarhice sau arborescente

-Se doreste crearea de obiecte din aceeasi familie de clase

39. Care este avantajul folosirii design pattern-ului Flyweight?

-Se pot crea structuri ierarhice

-Se reduce consumul de memorie

-Se pot aduce modificari obiectelor la runtime

40. Cand este folosit design pattern-ul Flyweight?

-Atunci cand se doreste modificarea functionalitatii unui obiect la runtime

-Atunci cand trebuie sa construim foarte multe instante ale unei clase, obiecte ce au o parte comuna

-Atunci cand este necesara crearea unei structuri ierarhice prin compunerea de obiecte

-Atunci cand se doreste adaugarea de noi functionalitati claselor existente

41. Flyweight este util atunci cand:

-Se doreste construirea unei singure instante

-Se doreste construirea unei structuri ierarhice

-Se doreste construirea a foarte multe obiecte ale unei clase, obiecte ce au o parte comuna

42. Care este asemanarea dintre Flyweight si Factory?

-Nodurile Composite pot fi privite ca Noduri frunza decorate

-Legaturile dintre clase

-Construirea de obiecte este gestionata de o clasa

-Nodurile Frunza pot reprezenta aceeasi instanta

43. Care dintre urmatoarele afirmatii despre Flyweight este adevarata?

-Implementarea lui este mare consumatoare de memorie

-Toate afirmatiile sunt adevarate

-Una din clase (FlyweightFactory) contine un HashMap pentru retinerea obiectelor asemanatoare

-Obiectele create sunt identice

44. Ce tip de design pattern este Proxy?

-Comportamental

-Structural

-Creational

45. Care este deosebirea dintre Proxy si Decorator?

-Proxy permite accesul la anumite functionalitati, iar Decorator adauga noi functionalitati

-Cele doua sunt identice

-Proxy modifica intreaga clasa, iar Decorator modifica comportamentul

46. Care este diferenta dintre Proxy si Adapter?

-Cele doua sunt complet diferite

-Ambele ascund, intr-un fel, clasa existenta

-Construirea de obiecte este gestionata de o clasa

-Legaturile dintre clase

47. Proxy este util atunci cand:

-Crearea de obiecte consuma multe resure

-Se doreste pastrarea functionalitatii unei clase ce se va realiza doar in anumite conditii

-Se doreste construirea de obiecte cu multe proprietati ce nu sunt obligatorii

48. Cand se utilizeaza Proxy?

-De fiecare data cand se doreste realizarea de permisiuni pentru toate obiectele

-De fiecare data cand se doreste realizarea de permisiuni pentru accesul la toate obiectele

-Nu folosim Proxy, deoarece nu este eficient

-De fiecare data cand se doreste realizarea de permisiuni pentru anumite obiecte sau pentru accesul la anumite functionalitati ale obiectelor

49. Participantii din cadrul State:

-O interfata + Clase concrete ce definesc starile obiectului + Clasa concreta ce defineste obiectul

-O interfata + Clasele concrete ce definesc obiecte in diferite stari

-Niciuna

-O interfata + Clase abstracte

50. Cand este folosit design pattern-ul Strategy?

-Atunci cand doreste adaugarea de noi functionalitati claselor existente

-Atunci cand avem mai multi algoritmi pentru rezolvarea unei probleme, iar alegerea implementarii se face la runtime

-Atunci cand este nevoie sa se utilizeze clase din framework-uri diferite

-Atunci cand se doreste modificarea functionalitatii unui obiect la runtime

51. Cand este folosit design pattern-ul State?

-Atunci cand este necesara crearea unei structuri ierarhice prin compunerea de obiecte

-Atunci cand un obiect isi schimba comportamentul pe baza starii in care se afla

-Atunci cand este nevoie sa se utilizeze clase din doua framework-uri diferite

-Atunci cand se doreste modificarea functionalitatii unui obiect la runtime

52. Participantii din cadrul Observer:

-interfate pentru observer si observabil + Clase concrete pentru observer si observabil

-O interfata pentru observer + O clasa concreta pentru observabil

-O interfata pentru observabil + O clasa concreta pentru observer

-Niciuna

53. Care este asemanarea dintre Proxy si Adapter?

-Ambele ascund, intr-un fel, clasa existenta

-Legaturile dintre clase

-Construirea de obiecte este gestionata de o clasa

-Cele doua sunt complet diferite

54. Participantii din cadrul Strategy:

-O interfata + Clase concrete ce implementeaza interfata + Clasa concreta ce contine o referinta de tipul interfetei

-O interfata + Clase concrete ce contin o referinta de tipul interfetei + Clasa concreta ce implementeaza interfata

-O interfata + Clase concrete ce contin o referinta de tipul interfetei

-Niciuna

55. Ce tip de design pattern este Observer?

-Structural

-Creational

-Comportamental

56. Care dintre urmatoarele afirmatii este adevarata?

-Observer permite instantierea unui singur obiect dintr-o clasa

-Observer defineste o relatie de 1:n

-Observer defineste o relatie de 1:1

-Toate

57. Prin ce se caracterizeaza design pattern-urile comportamentale?

-Controleaza relatiile complexe dintre clase

-Permit distributia responsabilitatilor pe clase si descrie interactiunea intre clase si obiecte

-Furnizeaza solutii pentru o mai buna interactiune intre obiecte si clase

-Toate

58. Care varianta de implementare a Singleton-ului presupune initializarea instantei la runtime, fara a intampina probleme la folosirea multithreading-ului?

-Thread Safe Singleton

-Lazy Initialization

-Eager Initialization

-Niciuna

59. Care dintre urmatoarele sunt design pattern-uri creationale?

-Singleton, Proxy, Facade

-Singleton, Prototype, Factory

-Adapter, Proxy, Facade

-Singleton, Proxy, Factory

60. Cu ce scop folosim design pattern-ul creational Prototype?

-pentru a crea o singura instanta a unei clase

-pentru gesiuntea unei familii de obiecte

-pentru a construi obiecte

-pentru a crea clone pentru obiectele a caror construire dureaza foarte mult sau consuma multe resurse

61. Este tip Singleton:

-Thread Safe Singleton

-Chain of Responsability

-Virtual Constructor

-Template Method

62. Builder poate fi corelat cu:

-Adapter

-Factory

-Singleton

-Prototype

63. Factory este util atunci cand:

-Se doreste construirea unei structuri ierarhice

-Este existenta o familie de obiecte intr-o aplicatie

-De fiecare data cand se doreste realizarea de permisiuni pentru toate obiectele

64. Care este relatia dintre Prototype si Decorator?

-Construirea unui obiect este centralizata

-Se pot stoca o serie de obiecte

-Se cloneaza obiectele si apoi se modifica

-Elementele de pe acelasi nivel pot fi clonate

65. Care dintre urmatoarele afirmatii este adevarata despre Builder?

-Presupune crearea unor obiecte complexe prin specificarea anumitor proprietati dorite din multitudinea existenta

-Are o singura varianta posibila de implementare

-Acest design pattern este folosit atunci cand crearea unui obiect dureaza foarte mult sau consuma resurse foarte multe

-Toate

66. Care dintre urmatoarele design pattern-uri creationale ajuta la crearea obiectelor complexe cu foarte multe atribute?

-Prototype

-Builder

-Factory

-Singleton

67. Cum se implementeaza Abstract Factory?

-Se creeaza un obiect iar pentru fiecare obiect exista cate o metoda

-Trebuie create noduri container si noduri frunza

-Trebuie sa fie unica metoda pentru a putea fi implementat

-Folosim abstractizare ci nu obiecte concrete pentru a implementa

68. Cati participanti sunt in design pattern-ul Prototype?

-2

-3

-4

-5

69. Care dintre urmatoarele reprezinta tipuri de implementare pentru Singleton?

-Eager initialization, Lazy Initialization, Thread Safe Singleton

-Eager initialization, Static initialization, Enum initialization

-Enum initialization, Inner class initialization

-Eager initialization, Lazy initialization, Static Singleton

70. Ce tip de design pattern ajuta la crearea unor obiecte concrete?

-Abstract Factory

-Factory Method

-Prototype

-Builder

71. Care design pattern ofera posibilitatea crearii de obiecte concrete dintr-o familie de obiecte?

-Factory

-Builder

-Singleton

-Niciuna

72. Reprezinta utilizare a design patternului Singleton:

-Existenta unei familii de obiecte intr-o aplicatie

-Crearea de view-uri pentru GUI

-Atunci cand obiectele create seamana intre ele, iar crearea unui obiect dureaza foarte mult sau consuma resurse foarte multe

-Deschiderea unei singure instanta ale unei aplicatii

73. Care sunt participantii din cadrul design patternului Prototype?

-Prototype Factory, Prototype

-Prototype, Virtual Prototype, Concrete Prototype

-Prototype, Concrete Prototype

-Prototype, Abstract Prototype, Concrete Prototype

74. Care sunt participantii din Design pattern-ul Factory?

-interfata/clasa abstracta, fabrica ce va crea obiectele concrete

-interfata/clasa abstracta, clasele concrete

-clasele concrete, fabrica ce va crea obiectele noastre

-interfata/clasa abstracta, clasele concrete, fabrica ce va crea obiectele concrete

75. Prototype este uril atunci cand:

-Doar in cazul in care nu exista clonare

-Ori de cate ori folosim clone()

-Atunci cand obiectele create nu se aseamana deloc

76. Care dintre urmatoarele afirmatii este adevarata despre design patternul Factory Method?

-Pentru apel se folosesc obiectele concrete

-Nu se foloseste enum, ci abstractizeaza nivelul de creare

-Sunt folosite structuri de tipul switch sau if-else pentru a alege ce obiecte vor fi create

-Are aceeasi participanti ca Simple Factory

77. Care este asemanarea dintre Prototype si Composite?

-Elementele de pe acelasi nivel nu pot fi clonate

-Elementele de pe acelasi nivel pot fi clonate

-Nu exista clonare

-Niciun element nu poate fi clonat

78. Ce reprezinta tipul de Singleton-Eager initialization?

-presupune faptul ca metoda nu o sa fie apelata de un alt fir de executie pana nu se termina metoda apelata deja pe un fir de executie

-initializarea instantei chiar daca aceasta nu e folosita

-cea mai implementata varianta Singleton

-contine o clasa imbricata in clasa Singleton

79. Ce reprezinta tipul de Singleton-Thread safe Singleton?

-cea mai implementata varianta Singleton

-initializarea instantei chiar daca aceasta nu e folosita

-contine o clasa imbricata in clasa Singleton

-presupune faptul ca metoda nu o sa fie apelata de un alt fir de executie pana nu se termina metoda apelata deja pe un fir de executie

80. Care sunt participantii in design pattern-ul Prototype?

-Prototype, ConcretePrototype

-Prototype, AbstractBuilder, AbstractPrototype

-Prototype, AbstractPrototype, ConcretePrototype

-Singleton, Prototype, AbstractPrototype

81. Care dintre urmatoarele sunt design patterns – creationale?

-Singleton, Builder, Factory

-Facade, Flyweight, Proxy

-Prototype, Strategy, Factory

-Builder, Adapter, Factory

82. O aplicatie de asigurari ofera,pe langa realizarea asigurarilor obligatorii, posibilitatea realizarii unei asigurari de sanatate, unei asigurari de calatorie si unei asigurari de pensie privata daca utilizatorul doreste acest lucru. Ce dessign pattern se regaseste in aceasta situatie?

-Factory

-Prototype

-Singleton

-Builder

83. Care este corelatia dintre Factory si Composite

-Se abstractizeaza nivelul de creare

-Prin intermediul fabricii sunt create nodurile container

-Fabrica este unica

-Nu exista corelatie

84. Ce rol are design patternul de tip Abstract Factory?

-Poate crea doar o familie de obiecte

-Se construieste cu ajutorul acestuia o structura ierarhica

-Nu poate fi clonat

-Introduce un nou nivel de abstractizare

85. Ce presupune Thread safe Singleton?

-Se poate apela metoda de pe un alt fir de executie chiar daca nu se termina metoda apelata deja pe un fir de executie

-Asigura faptul ca metoda nu o sa fie apelata de un alt fir de executie pana nu se termina metoda apelata deja pe un fir de executie

-Niciuna

-Toate

---

86. Cand poate fi implementat AbstractFactory?

-Introduce un nou nivel de simplificare

-Cand se creaza doua sau mai multe tipuri de obiecte fiecare avand o metoda

-Cand se doreste sa se adauge doar un obiect intr-o familie

-Toate

87. Care este diferenta dintre Factory Method si Simple Factory?

-In Factory Method nu se mai foloseste enum, ci abstractizeaza nivelul de creare

-In Factory Method se foloseste enum, nu se introduce un nou nivel de abstractizare

-Niciuna

-Nu exista diferente intre cele doua

88. Participantii design pattern-ului Prototype sunt:

-Interfata Prototype, Clase concrete, ConcretePrototype

-ConcreteProduct, AbstractProduct

-Interfata Prototype, ConcretePrototype

-AbstractProduct, ConcreteProduct, AbstractDecorator

89. Care dintre urmatoarele este considerat cel mai corect tip de Singleton?

-Thread Safe Singleton

-Eager initialization

-Lazy initialization

-Inner static helper class

90. Care este corelatia dintre Factory si Singleton?

-Fabrica poate fi unica

-Legaturile dintre clase

-Elementele se cloneaza

-Nu exista corelatie

91. Care design pattern asigura crearea unui singur obiect al unei anumite clase?

-Singleton

-Builder

-Proxy

-Prototype

92. Care dintre afirmatiile referitoare la Eager Initialization este corecta?

-Presupune initializarea instantei chiar daca aceasta nu este folosita

-Initializarea instantei se realizeaza doara daca aceasta este folosita

-Nu se initializeaza instanta niciodata

-Niciuna

93. Cum sunt definite design pattern-urile creationale?

-ajuta la initializarea si configurarea claselor si obiectelor

-ajuta la crearea familiilor de obiecte

-ajuta la compunerea si configurarea claselor si obiectelor

-controleaza relatiile complexe dintre clase

94. Care dintre urmatoarele este un tip de Singleton?

-Unic Initialization

-Thread Safe Initialization

-Singular Initialization

-List of Singleton

95. Care sunt participantii design patternului Prototype?

-Interfata si metoda de copiere si clonare

-Clasa concreta si interfata

-Clasele concrete

-Interfata si clasele de clonare

96. Factory este recunoscut dupa sintagma:

-O singura instanta

-Crearea de obiecte concrete

-Obiecte din aceeasi familie

-Crearea de clone pentru obiecte

97. Cum se poate corela design pattern-ul Factory cu design pattern-ul Singleton?

-obiectul poate fi unic

-clasa poate fi unica

-instanta poate fi unica

-fabrica poate fi unica

98. Care sunt participantii din cadrul design patternului Singleton?

-Concrete Singleton

-Abstract Singleton

-Singleton Factory

-Singleton

99. Care dintre afirmatii este adevarata in legatura cu Singleton?

-Nu presupune extinderea unei clase

-Singleton respecta principiile de programare orientata obiect

-Singleton este folosit pentru a implementa o clasa abstracta

-Un obiect Singleton nu poate fi trimis ca un parametru unei functii

100. Care dintre urmatoarele afirmatii este adevarata despre Factory?/

-Obiectele sunt create folosind o interfata comuna

-Este un design pattern creational

-Clientul are posibilitatea crearii de obiecte concrete dintr-o familie de obiecte, fara sa stie exact tipul concret al obiectului

-Toate

101. Care afirmatie referitoare la Abstract Factory este adevarata?

-Abstract Factory introduce un nou nivel de abstractizare

-Se construieste o structura ierarhica

-Implementarea lui este mare consumatoare de memorie

-Toate

102. Factory este folosit:

-pentru a crea obiecte dintr-o familie de clase

-atunci cand crearea unui obiect consuma foarte multe resurse. Se creeaza un prototip si este folosit pentru clonare

-pentru a crea o singura instanta pentru o clasa si are constructorul privat

-pentru a ajuta la crearea obiectelor complexe cu foarte multe atribute

103. Cand se utilizeaza Builder?

-Pentru construirea de obiecte simple fara atribute

-Pentru construirea de obiecte simple cu putine atribute

-Pentru construirea de obiecte complexe cu foarte multe atribute

-Toate

104. Care este rolul design pattern-lor creationale?

-Sunt concentrate pe cum sunt compuse clasele si obiectele pentru formarea de structuri complexe

-Ajuta la compunerea si configurarea claselor si obiectelor

-Furnizeaza solutii pentru o mai buna interactiune intre obiecte si clase

-Ajuta la initializarea si configurarea claselor si obiectelor

105. Ce presupune Enum Singleton?

-Asigura faptul ca metoda nu o sa fie apelata de o alta metoda

-Presupune utilizarea unei enumerari pentru crearea unica a instantei

-Este folosit pentru a usura workflow

-Contine o clasa imbricata in clasa Singleton

106. Care dintre urmatoarele afirmatii este adevarata despre Abstract Factory?

-Are ca participanti fabrica concreta si clasele concrete

-Simplifica interfata unei familii de clase

-Introduce un nou nivel de abstractizare fata de Factory Method

-Fiecare factory va crea un singur tip de obiect

107. Care dintre urmatoarele reprezinta un exemplu de utilizare pentru Singleton?

-Accesarea resurselor dispozitivelor mobile

-Conexiune unica la baza de date

-DocumentBuilderFactory din Android

-Toate

108. Care dintre urmatoarele design patternuri poate fi folosit atunci cand se doreste crearea de noi obiecte prin copierea unui obiect existent?

-Factory Method

-Builder

-Prototype

-Abstract Factory

109. Ce reprezinta Unit Testing?

-Este o metoda utilizata pentru a verifica daca aplicatia functioneaza conform cerintelor clientului

-Se refera la testarea individuala a unei componente, parti de cod, clasa sau metoda

-Toate

-Niciuna

110. Care dintre afirmatii este corecta?

-Fixture reprezinta un set de obiecte utilizate in test

-Unit Testing se realizeaza intr-un context bine definit in specificatiile de testare

-JUnit reprezinta o adaptare de la xUnit

-Toate

111. Care dintre urmatoarele metode de tip Assert nu se regaseste in JUnit?

-assertEquals

-assertSame

-Assert.IsNotNull

-assertTrue

112. Ce pattern permite încapsularea unui obiect in vederea controlării accesului la acesta?

-Template

-State

-proxy

-adapter

-momento

-strategy

-decorator

-niciunul dintre cele enumerate

-command

-observer

113. Pentru metoda următoare, care este numărul minim de teste unitare (cu diferite combinații de valori) ce trebuie scrise pentru a asigura un code coverage de 100%?

```
public int calcul(int a, int b){
    int rez = 0;

    if(a < 0)
        return 10;

    if(b < 0)
        rez = 20;
    else
        rez = 30;

    if(a >= 0 )
        rez += 5;

    return rez;
}
```

-2

-5

-6

-3

-4

-1

-nu poate fi asigurat un code coverage de 100%

114. Pentru metoda onMetoda() ce este correct implementata care dintre următoarele valori pot fi folosite pentru teste de tip Boundary (selectați unul sau mai multe răspunsuri)?

```
int oMetoda(int valoare) throws Exception {

    int rez = 0;

    switch (valoare) {

    case 1:
        rez = 10;
        break;

    case 5:
        rez = 20;
        break;
    case 10:
        rez = 30;
        break;

    default:
        throw new Exception("Valoare gresita");
    }

    return rez;
}
```

- Metoda nu poate fi testate prin teste Boundary, deoarece generează o excepție
- 30

- 0
- 20
- 1
- 5
- 10

115. Care dintre următoarele enunțuri descrie modelul Adapter corect?
- Nici una dintre aceste afirmații
- Acest model permite utilizatorului să adauge funcționalități noi la un obiect existent fără a-I modifica structura
- Acest model ascunde complexitatea sistemului/modulului și oferă clientului o interfață pe care o poate folosi pentru a accesa/utiliza sistemul
- Acest model este utilizat acolo unde trebuie să tratăm un grup de obiecte în mod similar ca un singur obiect
- Acest model permite definirea unei familii de algoritmi, fiecare fiind încapsulat într-un obiect. Clientul poate folosi orice algoritm fără să fie afectat de variația acestora
- Acest model este utilizat în principal pentru a reduce numărul de obiecte create și pentru a reduce amprenta de memorie

116. Dezvolți o soluție software pentru asociațiile de proprietari dintr-un cartier de blocuri care să le permită să gestioneze detaliile legate de costurile de întreținere aferente unei luni pentru fiecare proprietar de apartament. Aceste date (numărul apartamentului, costul apei calde, costul căldurii, costul gazelor, etc.) sunt stocate în fiecare lună. Soluția trebuie să permită salvarea acestor date la finalul lunii astfel încât să poată oferi în orice moment informații cu privire la istoricul acestor costuri pe fiecare apartament.
Independent de cerința anterioară, blocul are la intrare un interfon prin care este solicitat/controlat accesul. Deoarece copiii din cartier abuzează de acest serviciu și deranjează locatarii, se dorește adăugarea în sistem a unei camere video care să verifice dacă apelantul este adult sau copil. Această cameră va fi conectată la sistemul existent(nu se modifică interfonul) și va controla dacă se poate fi folosit interfonul sau nu. Dacă camera detectează un copil atunci interfonul nu va suna la apartamentul apelat.
Care sunt cele 2 design patternuri care rezolvă optim aceste două probleme?
-decorator
-chain of responsability
-state
-builder
-singleton
-command
-template
-strategy
-flyweight
-factory (simple sau method)
-observer

117. Dezvolți o soluție software pentru asociația de proprietari dintr-un bloc care să permită gestiunea eficientă a evenimentelor ce pot apărea, evenimente care necesită luarea unor măsuri adecvate și care să protejeze viața locatarilor. Soluția procesează diferite mesaje/evenimente primite de la senzori/sisteme din bloc și va lua următoarele decizii

- Dacă s-a declanșat alarma de incendiu atunci este notificat serviciul 112
- Dacă este anunțată o inundație atunci este notificat administratorul
- Dacă este doar o informare atunci sunt notificați locatarii
- (opțional) pentru alarme de incendiu și inundație sunt anunțați și locatarii

Independent de implementarea scenariului anterior găsiți o soluție care să permite integrarea sistemului existent de procesare a alertelor (cu interfața IProcesareEveniment) în sistemul național de alertare ce este construit în jurul interfeței IAlertarePublica.

Care sunt cele 2 design patternuri care rezolva optim aceste 2 probleme?

- Strategy
- Proxy
- Observer
- Singleton
- Template
- Adapter
- Flyweight
- Chain of responsability
- Factory (simple sau method)
- Command
- Memento
- State
- Decorator
- Composite
- Prototype

118. Pentru metoda oMetoda() ce este corect implementata care dintre următoarele valori pot fi folosite pentru teste de tip Range (selectați unul sau mai multe răspunsuri)?

```
int oMetoda(int valoare) throws Exception {

    int rez = 0;

    switch (valoare) {

    case 1:
        rez = 10;
        break;

    case 5:
        rez = 20;
        break;
    case 10:
        rez = 30;
        break;

    default:
        throw new Exception("Valoare gresita");
    }

    return rez;
}
```

- 1

- 0
- 20
- 5
- 30
- 10
- Metoda nu poate fi testată prin teste Range, deoarece generează o excepție

119. Luați în considerare următorul UnitTest. Ce se afișează după executarea sa?

```
public class TestMetode {

    @BeforeClass
    public void setUpBeforeClass() {
        System.out.print("Before ");
    }

    @Test
    public void test1() {
        System.out.print("Test1 ");
    }

    @Test
    public void test2() {
        System.out.print("Test2 ");
    }

    @After
    public void tearDown() {
        System.out.print("After ");
    }
}
```

- Before Test1 After Before Test2
- Before Test1 After Test2 After
- Before Test1 Test2 After
- Before Test1 Before Test2 After