1. in un piano per le attività di un progetto si trovano:

(a) solo i costi del personale

(b)solo diagrammi che indicano il lusso dei dati

(c) diagrammi dei casi d'uso, delle classi. degli stati

(d) tabelle diagrammi a barre e rete di attività.

2- con gestione di progetti si intende:

(a) la pianificazione e lo scheduling di attività

(b) tutte le altre risposte

(c) la gestione dei costi di produzione

(d) la scrittura di documentazione del software

3. il design pattern adapter permette a certe classi di interagire poiché:

(a) rende note le istanze di classi presenti

(b) elimina l'incompatibilità tra le interfacce.

(c) passa le chiamate tra classi implementate in linguaggi diversi

(d) tiene traccia dei nomi delle classi disponibili

4. e consigliabile usare facade:

(a) si vogliono far dipendere i client solo da interfacce

(b) se si vuole nascondere la complessità di un sottosistema.

(c) si hanno classi molto accoppiate

(d) si hanno classi con funzionalità simili

5. quale documento è generalmente usato come contratto tra cliente e fornitore:

(a) guida per l'utente del prodotto

(b) specifiche del software (srs)

(c) requisiti utente

(d) requisiti di sistema.

6. un documento dei requisiti per iscritto mira a fornire:

(a) valutazioni sul codice che deriverà dagli stessi requisiti

(b) informazioni da cui derivare la progettazione del sistema.

(c) dettagli sulla progettazione del sistema

(d) dati sul guadagno che si trarrà dopo la produzione

7. generalmente quale documento non è scritto per essere usato dai progettisti:

(a) requisiti di sistema

(b) i requisiti utente.

(c) specifiche del software (srs)

(d) nessuno dei documenti delle altre risposte

8. nella soluzione del design pattern state, ogni con concretestate:

(a) definisce l'interfaccia che il client usa

(b) definisce l'interfaccia associata ad uno stato

(c) mantiene l’istanza di state

(d) implementa un comportamento.

9.per realizzare il design pattern observer (of server), cosa è interessante osservare:

- lo stato del concretesubject.

10. l'uso della riflessione computazionale:

- permette ad un programma di usare più classi rispetto a quelle del codice.

11. la libreria che consente l'uso della riflessione in java fornisce:

- i metodi getdeclared methods() e getname.

12. la coesione di una classe è:

- alta sei tutti i modelli contribuiscono ad implementare un singolo compito.

13. in un sistema ad oggetto è buona norma sviluppare

(a) classi piccole, ciascuna responsabile per un piccolo compito.

(b) classi piccole, ciascuna contenente poco codice e molti dati

(c) poche classi, ognuna responsabile di più compiti

(d) classi grandi in modo da minimizzare l’accoppiamento

14. le leggi di lehman valgono:

(a) per qualunque sistema software

(b) solo per i sistemi di grandi dimensioni sviluppati da grandi aziende.

(c) per sistemi di grandi dimensioni sviluppati seconda il processo a spirale

(d) per sistemi di grandi dimensioni sviluppati da qualsiasi azienda

15. l'ingegneria del software è meglio descritta come la pratica:

(a) sviluppare sistemi di alte prestazioni per applicazioni di ingegneria

(b) sviluppare prodotti software tutti diversi tua loro

(c) sviluppare e mantenere software affidabili con un costo adeguato.

(d) sviluppare sistemi grandi assemblando prodotti poco efficienti/affidabili

16. la corrispondenza tra casi d'uso e requisiti:

- spesso si hanno le corrispondenze uno a molti e molti a uno.

17. sia la classe agent un singleton: agent a=agent.getistancet():

(c) verrà creata una sola istanza di agent (non importa il numero di volte che la linea di codice sarà seguita).

(d) si crea un’istanza di agent ogni volta che si incontra una

18. l factory method:

- la classe con concretecreator a un metodo che ritorna concreto concreteproduct.

19. in genere quali dei seguenti prende decisioni su istanziazioni di classe:

- singleton.

20. i settori di un ciclo a processo spirale sono:

(a) analisi, design, implementazione, integrazione e test

(b) analisi, design, implementazione, test e revisione

(c) identificare obiettivi, analisi rischi, sviluppo, revisione e pianificazione.

(d) revisione, decisione pianificazione, marketing

21. un processo software descrive:

(a) ciò che è in esecuzione per un certo programma

b) una certa modalità di funzionamento di un software

(c) le attività necessarie per lo sviluppo di un software.

(d) le attività che un software compie per trasformare dati in ingresso

22. i1 documento della specifica dei requisiti srs:

(a) una lista sommaria di cose che il software deve fare

(b) una lista precisa che il software deve fare.

(c) una stima delle risorse (tempo, costo, personale, etc.) richieste per costruire il software

(d) una descrizione precisa della struttura del sistema software

23. la relazione di ereditarietà è rappresentata in uml da:

(a) linee con un triangolo pieno ad una estremità

(b) linee con un triangolo vuoto e una estremità.

(c) linee con un rombo ad una estremità

(d) rettangoli annidati

24. nei diagrammi di interazione o collaborazione in uml:

(a) l'oggetto & un nodo di un grafo e le interazioni sono frecce

(b) gli oggetti sono allineati nella prima riga in alto

(c) l'interazione tra oggetti è una linea con sopra una freccia numerata.

(d) le classi sono nodi di un grafo e le relazioni tra classi sono frecce

25. lo sviluppo dei sistemi orientati agli oggetti incoraggia 1l'uso di dipendenze:

- da interfacce e non da classi.

26. tramite l'uso del factory method il cliente ha il vantaggio di:

- poter creare oggetti a runtime.

27. quali caratteristiche fondamentali a un sistema riflessivo:

- introspezione e intercettazione.

28. le caratteristiche che rendono il software differente da altri sono:

- complessità, modificabilità, invisibilità.

29. nella soluzione proposta dal design pattern mediator:

- una sottoclasse di mediatori chiama metodi di alcune superclassi.

30. nel design pattern state i clienti conoscono:

- nessuna sottoclasse di state.

31. le relazioni di aggregazione sono rappresentate in uml tramite:

- linee con un rombo ad una estremità

32. in uml, un diagramma delle attività mostra rettangoli:

- arrotondati per stati, i flussi con le frecce.

33. date le attività tl di durata 15 giorni, t2 di durata tre giorni, t3 durata sette giorni e sia: t3

dipendente da t2, tl e t2 eseguono in parallelo:

-qualsiasi ritardo su ti t2 o t3 si ripercuote sulla durata complessiva.

34. milestone e derivable possono essere:

- dei documenti di specifica di descrizione dell'architettura

35. quando si dovrà lavorare costantemente a contatto con i clienti è consigliabile il processo:

- spirale xp.

36. processo xp a:

- poca documentazione.

37. i processo evolutivo: lo sviluppo evolutivo

- è efficacemente applicabile a sistemi di piccole dimensioni

38. la legge di lehman sul cambiamento continuo dice:

- i sistemi dovrebbero essere continuamente cambiati per rimanere utili.

39. un metodo di tipo synchronized:

- aumenta il parallelismo del sistema.

40. un'interfaccia x, ovvero public interface x:

- ha lo scopo di definire un tipo.

41. un'interfaccia x ovvero public interface x:

- può essere istanziata.

42. i requisiti utente sono più spesso descritti tramite:

- linguaggio naturale, tabelle, diagrammi.

43. cosa significa che i requisiti devono essere completi e consistenti:

- significa che bisogna descrivere tutto e senza contraddizioni.

44. lo studio di fattibilità tiene in corrispondenza principalmente:

- obiettivi dell'organizzazione, costo del prodotto e tecnologie usate.

45. si abbina il frammento di codice: public libro getlibro() return b.getlib();:

- il codice è compatibile solo se b è stato inizializzato

46. i design pattern strutturali permettono di:

- comporre classi o oggetti.

47. uno dei vantaggi della riflessione computazionale consiste nella:

- possibilità di invocare metodi non conosciuti a design time.

48. i metodo new istance() è fornito dalla classe:

- class.

49. nel design pattern observer, si desidera ottenere:

- l'indipendenza tra un oggetto e il numero e tipo di osservatori

50. nel design pattern state i client conoscono:

- context.

51. l'attività tl dura 15 giorni, t2 dura cinque giorni e t3, t4 durano insieme 10 giorni. t3

dipende da t2 e ti, inoltre, t4 dipende da t3;

- il percorso critico è di 35 giorni.

52. date le attività ti di durata 15 giorni, t2 e t3 cinque giorni e sia t3 dipendente da t2e tl; ti e

t2 eseguono in parallelo:

- il percorso critico è di 20 giorni.

53. con correttezza del software si intende:

- aderenza allo scopo, conformità alle specifiche.

54. si riducono i tempi di sviluppo sei si:

- riusa codice ben documentato e corretto.

55. in uml, l'implementazione di un'interfaccia è rappresentata da:

- linee tratteggiate con un triangolo vuoto ad una estremità.

56. un diagramma di interazione o collaborazione è:

- un grafo che mostra le chiamate tra i metodi delle istanze.

57. nella sezione suggerita del design pattern classe adapter:

- adapter eredita adaptee.

58. nella sezione suggerita del design pattern classe adapter:

- elimina l'uso di una istanza all'interno di adapter

59. i design pattern:

(a) forniscono una collezione di oggetti

(b) -descrivono una soluzione riusabile ad un problema noto.

(c) forniscono un modo per riusare il codice

(d) forniscono una gerarchia di classi

60. i design pattern:

- sono. una soluzione collaudata ma generica.

61. la società moderna è:

- più indipendente da software rispetto a 15 anni fa.

62. il design pattern decorator può postare a:

- implementare classi grandi.

63. fissata un'interfaccia il design pattern decorator:

- fornisce una lista di istanze di classi che la implementano.

64. legge di lehman dall'aumento della complessità dice:

- la complessità tende a crescere con i cambiamenti.

65. nell'ingegneria dei requisiti è necessario:

- tutte le risposte risolvere i conflitti, revisionare i requisiti, priorità.

66. la navigabilità nelle relazioni dei diagrammi delle classi uml indica:

- per una classe quali altre classi conosce o usa.

67. le fasi di rup sono:

- avvio, elaborazione, costruzione, transizione, produzione.

68. il processo a cascata non è adatto a:

- produrre prototipi nella prima fase di sviluppo.

69. quali delle seguenti attività non è prevista da nessun processo software:

- valutare se la produzione del software è strategica.

70. grazie al singleton:

- fornire per ogni istanza un'identificazione unica.

71. tramite l'uso della riflessione computazionale un programma:

(a) rappresenta parte del programma stesso e agisce su sé stesso.

(b) legge il codice eseguibile dal programma stesso

(c) legge il codice sorgente dal programma stesso

(d) usa comunque solo le classi riferite nel codice sorgente

72. durante la pianificazione di un sistema si dovrebbero:

(a) progettare alcune parti del sistema

(b) usare diagrammi e lo standard uml

(c) stimare le durate delle attività che costituiscono lo sviluppo

(d) raccogliere i requisiti del sistema.

73. durante lo sviluppo è possibile organizzare:

(a) task secondo percorsi paralleli se ci sono indipendente.

(b) task di breve durata se si hanno tanti sviluppatori

(c) task di lunga durata se si hanno pochi sviluppatori

(d) task dipendenti tra loro se si hanno tanti sviluppatori

74. per il design pattern observer le sottoclassi di obeserver:

(a) dovrebbero essere al massimo 20

(b) dovrebbero avere almeno 2 istanze ciascuna

(c) dovrebbero essere almeno due.

(d) potrebbero avere una sola istanza ciascuna

75. nel design pattern mediator:

(a) collague conosce la sua sottoclasse

(b) mediator conosce la sua sottoclasse

(c) colleague conosce la sottoclasse di mediator

(d) collauge conosce mediator.

76. un caso d'uso rappresenta: ù

(a) la struttura del sistema

(b) il comportamento del sistema dal punto di vista delle interazioni con l'utente.

(c)l’insieme delle interazioni tra parti interne al sistema

(d)un insieme di obiettivi del sistema

77. usando mediator la complessità tra classi è ridotta perché:

- ciascuna classe non dovrà gestire interazioni con altre classi.

78. design pattern state fa sì che:

- un oggetto sembri cambiare la sua classe quando il suo stato varia.

79. il metodo join permette:

- far ripartire il thread su cui si invoca.

80. cosa sono i requisiti funzionali e non funzionali:

- indicano cosa il sistema deve fare ed alcuna qualità.

81. per i test xp i test funzionali sono progettati o scritti:

- dagli sviluppatori

82. in genere la pianificazione viene svolta:

- subito dopo la progettazione del sistema.

83. il coupling tra classi è:

- alto se il codice della classe e facile da comprendere.

84. un ingegnere del software che si occupa in modo responsabile può:

- leggere i dati confidenziali.

85. quale diagramma uml si può inserire in un documento dei requisiti:

- diagramma delle classi.

86. quale documento è usato come contratto tra cliente e fornitore:

- requisiti di sistema.

87. nei documenti dei requisiti le dipendenze mirano ad indicare:

- l’origine dei requisiti, le relazioni e funzionalità.

88. le dipendenze tra un'applicazione e le piattaforme hardware e software:

- sono riducibili studiando certe soluzioni.

89. il design pattern composite ha come obiettivo:

- far apparire in modo uniforme i client sono oggetti semplici e composti.

90. si usa il facade quando: è consigliabile usare il facade quando:

- si vuole nascondere la complessità di un sottoinsieme.

91. i metodo getfields():

- si può invocare su una istanza di classe.

92. cosa fa il *current thread*()

(a) restituisce un identificativo del thread

(b) sospende il thread corrente

(c) indica se il thread su cui viene invocato è running

(d) indica se il thread su cui viene invocato è stato avviato

93. una variabile di un certo tipo t

(a) può contenere solo istanze di t

(b) può contenere istanze di t e di supertipi di t

(c) può contenere qualunque istanza

(d) può contenere istanze di t e di sottotipi di t

94. ln genere per convalidare i requisiti non è possibile

(a) eseguirli tramite dei test

(b) servirsi di prototipi

(c) usare metodi di analisti di consistenza

(d) sviluppare dei test

95.la frase “il sistema dovrà fornire prestazioni adeguate” è:

(a) un requisito molto importante

(b) un requisito non funzionale

(c) un requisito funzionale

(d) un requisito interpretabile in tanti modi

96. per il design pattern observer, cosa interessante osservare?

(a) ll comportamento del subject

(b) ll comportamento del concreteobserver

(c) lo stato del conereteobserver

(d) lo stato del concretesubject

97. (a) stati sequenziali

(b) gerarchie tra stati

(e) stati composti

(d) stati concorrenti

98. le leggi di lehman si adattano a:

(a) prodotti software grandi

(b) prodotti software piccoli

(c) prodotti software con cots

(d) piccole aziende di software

99. i test su un singolo componente software sono detti:

(a) gamma test

(b) beta test se con i dati del cliente

(c) unit test

(d) alpha test

100. un …. di un caso d’uso serve a rappresentare:

(c) una specializzazione di un caso d’uso di base

101. per design pattern chain of responsability:

(a) i possibili riceventi sono più di uno //la classe ricevente potrebbe avere un singolo metodo pubblico

(b) le classi richiedenti devono essere più di una

(c) si richiede di implementare tanti metodi per ciascuna classe richiedente

(d) si ha che il numero di richiedenti è pari al numero di riceventi

102. come va catalogato in termini di ambito e tipologia il design pattern prototype?

(pattern creazionale)

(a) class creational object behavior

103. la legge di demeter dice:

una classe non dovrebbe conoscere i dettagli implementativi di un’altra classe*.*

104. un metodo di una classe che chiama un metodo astratto (provoca un errore)

(esegue un momento esecuzione)

105. per il design pattern decorator, generalmente il riferimento al concreatecomponent

106. per il design pattern adapter, il ruolo adapter

(a) è una superclasse di altre classi del design pattern

(b) è una classe

(c) è una classe di libreria

(d) è un'interfaccia

107. la classe che ha il ruolo facade ha

(a) una sola classe client

(b) nessuna classe client, difatti il facade potrebbe avere il metodo main

(c) un certo numero di classi client

(d) una sola classe nel sottosistema

108. una classe con tante linee di codice e tante responsabilità

(a)è molto complessa

(b) serve ad avviare operazioni su altre classi

(c) contribuisce a diminuire la complessità di altre classi

(d) serve a creare istanze di oggetti

109. la soluzione per il design pattern composite avrà tipicamente

(a) una istanza per il ruolo leaf e varie istanze per composite

(b) una istanza per ciascun ruolo composite

(c) una istanza per il ruolo composite e varie istanze per leaf

(d) una istanza per ciascun ruolo leaf e composite

110. in uml, un oggetto può essere rappresentato da

(a) un rettangolo contenente il nome dell'oggetto, ":" ed il nome della classe a cui appartiene

(b) un rettangolo con bordi arrotondati contenente il nome dell'oggetto

(c) un rettangolo contenente "object" ed il nome della classe

(d) un rettangolo contenente il nome della classe

111. il design pattern observer consiste di

(a) almeno una istanza di observer

(b) un subject ed un certo numero di observer

(c) due subject e due observer

(d) subject con la sua sottoclasse, e observer con almeno una sottoclasse

112. le convenzioni di codifica consistono di

(a) degli standard per la scrittura del documento srs

(b) delle leggi sul copyright

(c) regole per la scrittura del singolo requisito

(d) norme per scegliere i nomi di metodi e classi

113. il termine spaghetti code indica

(a) un metodo che fa tante cose

(b) una classe cha ha tanti metodi

(c) codice creato per il cliente

(d) la progettazione ad oggetti

114. con evoluzione del software si intende

(a) l'esecuzione di test su un prodotto non ancora consegnato (un processo di sviluppo che apporta continue modifiche al prodotto prima del rilascio

(c) la modifica di un prototipo non ancora consegnato

(d) il cambiamento del sistema software dopo la sua consegna

115. si abbia il frammento di codice: public box getbox() { return new minibox (); }

(a) il frammento sta svolgendo il ruolo di un singleton

(b) ll codice è compilabile solo se minibox è sottoclasse di box

(c) ii frammento sta svolgendo il ruolo di un product

(d) il codice dovrebbe essere all'interno di una superclasse

116. l'obiettivo del mediator è di avere

(a) varie classi che non conoscono il mediator

(b) interazioni distribuite su più classi

(c) interazioni per lo più isolate su una classe

(d) classi che si chiamano fra loro

117. per il design pattern bridge, la classe abstraction

(a) tiene il riferimento di tante istanze

(b) chiama solo i metodi di una o più classi di libreria

(c)tiene il riferimento di una istanza

(d) dipende da una o più classi di libreria

118. per il design pattern decorator, il ruolo decorator

(a) tiene un riferimento

(b) è un'interfaccia

(c) è una classe astratta

(d) può essere un'interfaccia

119. per eliminare le istruzioni condizionali

(a) si potrebbe usare il design pattern state

(b) si potrebbe usare il design pattern singleton

(c) si possono inserire le precondizioni

(d) si potrebbe usare il design pattern facade

120. quale di questi è un valido motivo per usare il pattern prototype?

(a) per fa sì che il sistema sia indipendente dal tipo di prodotto concreto che usa

(b) per offrire operazioni di alto livello, da comporre su operazioni primitive

(c) per implementare il supporto alle macro

(d) per supportare l'undo, ripercorrendo il log e annullando le modifiche a ritroso

121. il ruolo state, per il design pattern state:

(a) permette ad un’istanza di cambiare comportamento

(b) metodo che usano variabili di stato

(c) permette di avere una variabile che tiene un’istanza di classi diverse.

(d) ha lo scopo di creare istanze

122. il design pattern state permette in vari casi di:

(a) spostare istruzioni condizionali, ma non ridurlo o eliminarle

(b) tenere lo stato di un oggetto

(c) eliminare o ridurre inesattezze

(d) eliminare istruzioni condizionali

123. nel singleton, il metodo statico serve

(a) ad avere una sola istanza

(b) a dare ai client la possibilità di invocare un metodo su una classe

(c) a violare ai client l'uso del new sul singleton

(d) ad affidare ad un unico oggetto la responsabilità di creare istanze

124. in un diagramma uml di interazione, le interazioni sono rappresentate da:

(a) frecce

(b) lince con sopra frecce

(c) lince con triangoli ad una estremità

(d) frecce piene

125. ua metodo di una classe che chiama un metodo astratto

(a) esegue codice che è determinato al momento dell'esecuzione

(d) provoca un errore, rilevato al momento della compilazione

(c) i metodi astratti non esistono

(d) provoca un errore, rilevato al momento dell'esecuzione

126. per il processo xp

(a) il codice prodotto ha bassa qualità

(b) la documentazione ed il codice sono aggiornati continuamente

(c) la pianificazione iniziale delle attività è molto precisa

(d) la documentazione è molta

127. la legge di demeter dice

(a) non chiamare, aspetta di essere chiamato

(b) non sviluppare classi enormi

(c) parla solo con gli amici

(d) non parlare solo con gli amici

128. per il design pattern bridge, implementor

(a) è un'interfaccia

(b) può essere un'interfaccia

(c) è una classe astratta

(d) tiene un riferimento

129. un vantaggio offerto dal design pattern facade è:

(a) ridurre le dipendenze di compilazione dei sistemi

(b) ridurre il numero di classi

(c) poter avere tante funzionalità diverse

(d) lasciar interagire oggetti direttamente

130. il design pattern observer, il subject

(a) chiama un metodo e non passa parametri a tale metodo

(b) implementa un metodo update

(c) invia il riferimento a sé stesso

(d) è una classe astratta o un'interfaccia

131. il pattern adapter può essere usato per

(a) una gestione di un object pool

(b) la riduzione dell'accoppiamento tra classi

(c) decide quale oggetto istanziare

(d) controllare il verificarsi di condizioni prima di chiamare metodi.

132. le pratiche dell'ingegnere del software sono utili per produrre

(a) una singola versione di un sistema software

(b) tante versioni di un sistema software, in un arco di tempo lungo

(c) tanti sistemi software che derivano da sistema software preesistenti

(d) un preciso numero di versioni di un sistema software

133. processo basato su cots include

(a) modifica (evoluzione) di un sistema software

(b) acquisizione di componenti, modifiche su questi ed integrazione

(c) molte interazioni tra le persone (clienti, utenti, sviluppatori)

(d) sviluppo di componenti da riusare in successivi prodotti

134. quali delle seguenti non è una caratteristica del pattern prototype

(a) il prototipo è istanziato esternamente e poi passato come parametro

(b) ha una potenziale perdita di efficienza per l'aumento del numero delle classi

(c) se clono un prototipo in modalità challow le copie potrebbero interferire l'una con l'altra

(d) quando creazione di nuove istanze è costosa, clonando migliora la performance

135. per il design pattern composite, la superclasse

(a) definisce i metodi che i client possono chiamare sulle sottoclassi

(b) definisce solo metodi indipendenti dalle funzionalità dell'applicazione

(c) chiama alcuni metodi su istanze delle sottoclassi

(d) contiene metodi che sono chiamati dalle sottoclassi

1. usando la programmazione funzionale di java, data una lista pl di istanze di Persona (codice sotto), implementare una classe con

A) un metodo che trova nella lista PL le persone che hanno nazione uguale a un parametro passato in ingresso al metodo, e restituisce la lista dei nomi delle persone trovate;

B) Un metodo che conta il numero di nazioni diverse presenti nella lista PL

C) Un metodo che crea una lista di istanze di Persona, aventi i dati delle istanze presenti in PL eccetto per la nazione che dovrà contenere solo la prima lettera della nazione preesistente

D) Un metodo che costituisce la lista delle età della lista PL.

public PL (String Nome, String Nazione, int eta){

A) Un metodo che trova nella lista PL le persone che hanno nazione uguale a un parametro passato.

public static List<String> getNomiByNazione(List l, String Nazione)

{

Stream<PL> lstream = l.stream();

List<String> PL = lstream

.filter(s->s.getNazione().equals(Nazione))

.map(PL::getNome)

.collect(Collectors.toList());

return PL; }

B) Un metodo che conta il numero di nazioni diverse presenti nella lista PL

public static long getNumeroNazioni(List l)

{

Stream<Persona> lstream = l.stream();

long n = lstream

.map(Persona::getNazione)

.distinct()

.count();

return n; }

C) Un metodo che crea una lista di istanze di Persona, aventi i dati delle istanze presenti in PL eccetto per la nazione che dovrà contenere solo la prima lettera della nazione preesistente

public static List<Persona> getPersonaObbreviato(List l)

{

Stream<Persona> stream1 = l.stream();

List<Persona> persone1 = stream1

.map(s -> new Persona(s.getNome(), s.getNazione().substring(0,1), s.getEta()))

.collect(Collectors.toList()); }

D) Un metodo che costituisce la lista delle età della lista PL.

public static List<Integer> getListaEta(List l)

{

Stream<Persona> lstream = l.stream();

List<Integer> PL = lstream

.map(Persona::getEta)

.distinct()

.sorted((o1, o2) -> o1.compareTo(o2))

.collect(Collectors.toList());

return PL; }

1. Data una Lista Car di istanze Auto, implementa una classe con

A) Un metodo che ritorna una lista contenente i costi delle istanze di Auto nella lista car

B) un metodo che restituisce il numero di istanze che hanno colore rosso e costo 1000

C) Un metodo che sono i costi delle prime 10 istanze della lista car

D) Un metodo che restituisce il costo massimo presente nella lista car

Public record Auto (String categoria, String colore, int costo, String modello) {

}

A) Un metodo che ritorna una lista contenente i costi delle istanze di Auto nella lista car

public static List <Integer> getCosto (List car)

{

Stream<Auto> carstream = car.stream(); //carstream è il nome

List<Integer> Costi = carstream

.map(Auto:: getCosto)

//.distinct()

.collect (Collectors.toList());

return Costi;

}

B) Un metodo che restituisce il numero di istanze che hanno colore rosso e costo 1000

public static Long getNumeroIstanza (List car)

{

Stream<Auto> carstream = car.stream();

Long N = carstream

.filter(s-> s.getColore().toLowerCase().equals("rosso") )

.filter(s-> s.getCosto() == 1000)

.count();

return N;

}

C) Un metodo che restituisce i costi delle prime 10 istanze della lista car

public static List<Integer> getPrimiDieci (List car)

{

Stream<Auto> carstream = car.stream();

List<Integer> Auto10 = carstream

.limit(10)

.map(Auto:: getCosto)

.collect(Collectors.toList());

return Auto10;

}

D) Un metodo che restituisce il costo massimo presente nella lista car

public static Auto getCostoMax (List car)

{

Stream<Auto> carstream = car.stream();

Auto cm = carstream.max(Comparator.comparing(Auto::getCosto)).get();

return cm;

1. Usando la programmazione funzionale di Java, data una lista PL di istanze di Persona

(codice sotto), implementare una classe con

A) Un metodo che prende in input due parametri

B) Un metodo che calcola la somma delle età di PL

C) Un metodo che restituisce mappa contente coppie (…)

Map

D) Un metodo che restituisce una lista di string contenente le nazioni, senza ripetizioni presenti

nella lista.

Public record Persona (String nome, String nazione, int (età) {

}

1. Usando la programmazione funzionale di Java, data una lista t di istanze di Triang, implementare una classe con:

(a) un metodo che restituisce le istanze della lista t che sono triangoli (suggerimento: verificare che la somma degli angoli sia 180).

(b) un metodo che restituisce le istanze di Triang che sono triangoli rettangoli (suggerimento: un angolo è 90 e la somma degli angoli è 180),

(c) un metodo che restituisce il valore maggiore dei perimetri delle istanze di t,

(d) un metodo che restituisce la somma dei perimetri delle istanze di t.

public record Triang(int latol, int lato2, int lato3, int angolol, int angolo2, int angolo3) {

}

1. Usando la programmazione funzionale di Java, data una lista f di istanze di Figura, implementare una classe con:

(a) un metodo che restituisce le istanze f che sono quadreti o rettangoli (suggerimentose i quattro angoli sono uguali).

(b) un metodo che restituisce una lista contetente il valore del lato più piccolo di ciascuna istanza in f

(c) un metodo che restituisce il valore minore dei perimetri delle istanze in f,

(d) un metodo che restituisce la somma dei perimetri delle istanze in f.

public record Figura(int latol, int lato2, int lato3, int lato4, int angolol, int angolo2, int angolo3, int angolo4) {

}

Per il seguente codice: (x2)

Public interface Esame {

Public void Persona ();

Public void registra ();

}

Public class Nuovo implementsEsame {

Public void Prenota () {

System.out.println (“Presentazione Ok”);

}

Public void registra ()

{

System.out.println (“Non è possibile registrare l’esame”);

}

}

Public class Prenotato implementsEsame {

Public void Prenota () {

System.out.println (“Non è possibile prenotare l’esame”);

}

Public void registra () {

System.out.println(“Registrazione Ok”);

}

}

A) completare l'implementazione del design Pattern State con almeno due classi

1.

public class Context { //context

private Esame esame;

public Context(){

esame= null;

}

public void setEsame(Esame esame){

this.esame = esame;

}

public Esame getEsame(){

return esame;

}

}

2. Per il seguente codice: (x2)

A) completare l'implementazione del design Pattern State con almeno due classi

public class EsamePatternDemo { //state Pattern Demo

public static void main(String[] args) {

Context context = new Context();

Nuovo N = new Nuovo();

N.prenota(context);

System.out.println(context.getEsame().toString());

Prenotato P = new Prenotato();

P.registra(context);

System.out.println(context.getEsame().toString());

}

}

B) Inserire in una delle classi la memorizzazione del nome della materia e della data dell'esame

private String Data;

public void setData(String d) {

this.Data = d;

}

public String getData() {

return Data;

}

private String Materia;

public void setMateria(String m){

this.Materia = m;

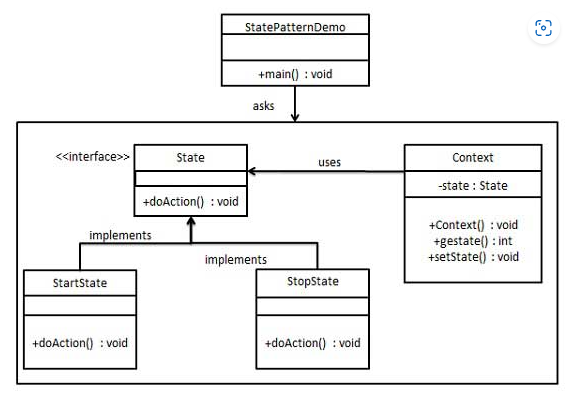
}

public String getMateria(){

return Materia;

}

C) dare il nome dei ruoli per tutte le classi e interfacce del design Pattern.

D) disegnare il diagramma UML delle classi. 

E) Disegnare il diagramma UML di sequenza illustrando l'esecuzione a partire da una classe

appropriata.

1. Per il seguente codice:

(a) indicare il design pattern, il ruolo e il nome tipico dato al metodo,

(b) implementare la classe C1 che usa in modo appropriato al design patter la classe data

(c) implementare una variante nota dello stesso design pattern,

(d) disegnare il diagramma UML della classe fornita e della classe C1,

(e) disegnare il diagramma UML di sequenza per la classe data e la classe C1, con l’esecuzione che parte dalla classe C1

Public class ContatoreBeep {

Private static ContatoreBeep c=new ContatoreBeep();

private int contatore;

private ContatoreBeep(){

contatore=0;

}

Public void printx(){

System.out.println(“x”+contatore);

contetore++;

}

Public static ContatoreBeep metodo1(){

Return c;

}

Public void reset(){

Contatore=0;

}

Public void incr(){

Contatore++;

}

}

2. Per il seguente codice:

(a) indicare il design pattern ed i nomi dei ruoli di ciascuna interfaccia e classe fornita,

(b) indicare il nome appropriato di Tipol, Tipo2 e parola chiave mod1,

(c) implementare il codice appropriato di una classe C1 che chiama i metodi su un'istanza di Mini,

(d) disegnare il diagramma UML delle classi fornite e della classe C1 (indicando nomi classi, nomi metodi, e relazioni fra classi),

(e) disegnare il diagramma UML di sequenza in cui l'esecuzione inizia dalla classe C1

public interface Busta {

public String getTipo ();

public int getPeso ();

}

public class Grande implements Busta {...}

public class Mini implements Busta {...}

public class Deposito {

public Tipol getBusta () {

return new Grande();

}

public Tipo2 getMini () {

return mod1 Mini();

}