Warsztat

OPIS

Twoim zadaniem będzie stworzenie aplikacji dla warsztatu samochodowego. Warsztat przyjmuje samochody do naprawy. W pierwszej kolejności ocenia czy dana marka samochodu jest przez niego obsługiwana. Następnie identyfikuje uszkodzone części i przeprowadza ich wymianę. Jeżeli okażę się, że części nie ma wystarczająco na warsztacie, to odstępuje od naprawy samochodu wystawiając rachunek na 50 zł (koszt diagnozy problemu). Jeżeli części wystarczy, to wykonywana jest naprawa. Każda część kosztuje inaczej, a koszt naprawy, to suma kosztu wszystkich części pomnożona przez 131%.

CZEŚĆ 1.

---

1. W pierwszym punkcie przygotujesz klasy niezbędne do implementacji aplikacji warsztatu. Wykonaj następujące kroki:

---

a) Utwórz pakiet `sda.java9.cons.cars`. Kolejne klasy będziesz tworzyć w tym pakiecie
b) Utwórz klasę publiczną `Warsztat`. Będzie to główna klasa aplikacji
c) Utwórz klasę publiczną `Samochod`. Będzie to klasa reprezentująca samochód do naprawy
d) Utwórz klasę publiczną `Czesc`. Będzie to klasa reprezentująca część samochodu, abstrakcyjną
e) Utwórz klasę publiczną `Silnik`
f) Utwórz klasę publiczną `Zawieszenie`
g) Utwórz klasę publiczną `Hamulce`
h) Utwórz klasę publiczną `Karoseria`
i) Utwórz klasę publiczną `SkrzyniaBiegow`

---

2. Teraz nadamy wszystkim klasom sens, a więc wypełnimy je właściwościami (polami)

---

a) W klasie `Samochod` dodaj pola:
   i) `marka` (tekst)
   ii) `rok` (liczba)
   iii) `popsuteCzesci` (tablica lub lista/zbiór obiektów klasy `Czesc`)
b) W klasie `Czesc` dodaj pola:
   i) `nazwa` (tekst)
   ii) `koszt` (`double` lub `BigDecimal` [preferowany] w skalowaniu do dwóch miejsc po przecinku)
c) W klasie `Warsztat` dodaj pola:

i) `obslugiwaneMarki` (tablica lub lista/zbiór obiektów klasy `String`)
ii) `czesciNaStanie` (tablica lub lista/zbiór obiektów klasy `Czesc`)
iii) `zysk` (`double` lub `BigDecimal` [preferowany] w skalowaniu do dwóch miejsc po przecinku)

---

3. Część informacji, którą będzie wykorzystywać aplikacja są globalne, więc stworzymy dla nich odpowiednie pola statyczne.

---

a) W klasie `Samochod` utworz pole statyczne:
   i) `WSZYSTKIE_MARKI` (tablica lub lista/zbiór obiektów klasy `String`) !pole powinno być publiczne
b) W klasie `Warsztat` utwórz pole statyczne:
   i) `KOSZT_BEZ_WYMIANY` (`double` lub `BigDecimal` [preferowane] w skalowaniu do dwóch miejsc po przecinku) !pole powinno być prywatne
   ii) `MARZA` (`double` lub `BigDecimal` [preferowane] w skalowaniu do dwóch miejsc po przecinku) !pole powinno być prywatne

---

4. Utworzonym polom statycznym nadamy odpowiednie wartości. Spraw też, aby nie dało się ich potem nadpisać.

---

a) Polu `KOSZT_BEZ_WYMIANY` przypisz wartosc `50.0` lub `new BigDecimal("50.00", 2)`
b) Polu `MARZA` przypisz wartość `31.0` lub `new BigDecimal("31.00", 2)`
c) Polu `WSZYSTKIE_MARKI` przypisz tablicę typu `String` o rozmiarze 8 i wymyśl 8 marek, które chcesz w niej umieścić. Możesz wykorzystać do tego celu wyrażenie inicjalizujące tablicę typu `new String[] {"a", "b", "c"}` lub statyczny blok inicjalizacyjny.

---

5. Przed Tobą trochę żmudnej roboty. Do wszystkich klas, w których dodałeś pola (pola instancyjne, nie pola statyczne) utwórz metody typu GET i SET

---

6. Zadbamy teraz o części do Warsztatu.

---

a) Spraw, aby klasa `Czesc` była abstrakcyjna.
b) Klasy `Silnik`, `Zawieszenie`, `Hamulce`, `Karoseria` i `SkrzyniaBiegow` powinny dziedziczyć po klasie `Czesc`
c) W klasie `Czesc` dodaj konstruktor przyjmujący parametr `nazwa` oraz ustawiający to pole o tej samej nazwie. Pole `koszt` niech pozostanie nieustawione.

d) Do każdej z klas z punktu wyżej (tych, które dziedziczą po `Czesc`) napisz konstruktor domyślny. W konstruktorze wywołaj konstruktor nadklasy, korzystając z instrukcji `super` i przekaż jako parametr odpowiednią nazwę części. Ustaw również pole `koszt` wymyśloną przez siebie wartością. Postaraj się tylko, aby pamiętać o max dwóch miejscach po przecinku, więc "111,11111" odpada ;)

---

7. W każdej z klas konkretnych części nadpisz metodę "String toString()", tak aby metoda ta zwracała obiekt klasy String reprezentujący stan bieżącego obiektu. Masz do dyspozycji dwa pola: nazwa i koszt, więc wiele możliwości nie ma :)

---

8. Teraz trzeba zatroszczyć się o warsztat i sprawdzić jak to wszystko razem wygląda, na tym etapie

---

a) W klasie `Warsztat` napisz konstruktor, który przyjmie dwa parametry: tablicę obiektów String reprezentującą obsługiwane marki; tablicę obiektów `Czesc` reprezentującą zestaw części na warsztacie. Oczywiście w konstruktorze przypisz parametry do odpowiednich pól.
b) Nadpisz metodę `String toString()`, aby klasa `Warsztat` potrafiła przedstawić swoją reprezentację w postaci obiektu klasy `String`. Powinieneś uwzględnić w wyniku tej metody stan wszystkich pól klasy `Warsztat`. Żeby wyświetlić zawartość pól `obslugiwaneMarki` i `czesciNaStanie` możesz skorzystać z metody statycznej `toString` w klasie `Arrays` (pod warunkiem, że nie użyłeś listy/zbioru, a tablicę).
c) Napisz metodę `public static void main(String[] args)`, aby dało się uruchomić klasę `Warsztat`
d) W metodzie `main` utwórz tablicę typu `String` zawierającą zestaw obsługiwanych marek oraz tablicę typu `Czesc` zawierającą zestaw części na warsztacie.
e) Na podstawie utworzonych tablic stwórz nowy obiekt klasy `Warsztat` i wyświetl jego reprezentację tekstową w oknie konsoli.

---

9. Mamy podstawy i w tym punkcie każda klasa powinna być gotowa do dalszego rozwoju. Teraz dodamy kilka szczegółów:

---

a) W klasie `Warsztat` napisz metodę statyczną, która zwraca tablicę typu `String`. Metoda powinna wybierać losowo obsługiwane marki samochodów. Skorzystaj z pola statycznego `WSZYSTKIE_MARKI` w klasie `Samochod`, aby wybrać tylko istniejące marki. Zadbaj, aby w tablicy nie było powtórzeń. Wykorzystaj tą metodę do tworzenia tablicy marek dla klasy Warsztat (punkt 8.d i 8.e)

b) W klasie `Warsztat` napisz metodę statyczną, która zwraca tablicę typu `Czesc`. Metoda powinna tworzyć losową liczbę obiektów klasy `Czesc`. Pamiętaj, że prawdziwe obiekty są konkretnych klas dziedziczących po klasie `Czesc`. Zarówno ilość części powinna być losowa (z przedziału od 10 do 100), ale również jakie części będą tworzone też powinno być losowe. Wykorzystaj tą metodę do tworzenia tablicy części dla klasy `Warsztat` (punkt 8.d i 8.e)
c) Utwórz 5 obiektów klasy `Warsztat` i zainicjalizuj je losowym zestawem obsługiwanych marek i losowym zestawem posiadanych części. Wyświetl wszystkie te obiekty w oknie konsoli.

CZĘŚĆ 2

Posiadamy już w naszym programie Warsztat, który obsługuje konkretne marki samochodów oraz posiada zestaw części do ich naprawy. Teraz zadbamy o to, aby pojawiły się samochody i aby coś się w nich mogło popsuć.

---

10. Twoim zadaniem będzie dostarczyć teraz narzędzie do efektywnego tworzenia samochód, z którymi Klienci przychodzą do warsztatu. Zaczniemy od przypadku „liniowego" gdzie samochód ma popsuty silnik.

---

a) W klasie Samochod dodaj metodę statyczną o sygnaturze public static Samochod stworzLosowySamochodZPopsutymSilnikiem().
b) Wewnątrz metody stwórz nowy obiekt klasy Samochod korzystając z konstruktora domyślnego.
c) Przypisz losową markę do samochodu, wybierając jedną z dostępnych marek zdefiniowanych w polu statycznym WSZYSTKIE_MARKI
d) Przypisz losowy rok produkcji samochodu wybierając z przedziału od roku 1990 do roku 2018.
e) Utwórz nowy obiekt klasy Silnik i przypisz go do tablicy popsutych części.
f) Na końcu zwróć poprawny obiekt klasy Samochód. Powinien mieć ustawione pola rokProdukcji oraz marka, a także zainicjalizowaną tablicę części zawierającą pojedynczy obiekt klasy Silnik.

---

11. Skoro przeszliśmy przez dosyć statyczny przykład niesprawnego samochodu, to teraz stwórzmy bardziej użyteczną metodę fabrykującą.

---

a) W klasie Samochod dodaj metodę statyczną o sygnaturze public static Samochod stworzLosowySamochod ().
b) Skorzystaj z punktów b), c) i d) z poprzedniego punktu
c) Wybierz teraz losową liczbę z przedziału od 1 do 5 (równego liczbie klas dziedziczących po klasie Czesc)

d) Dodawaj losowo wybraną część do samochodu. Losuj tyle części ile równa jest wartość z punktu c). Pamiętaj tylko, aby nie dodać do samochodu części tego samego typu więcej niż jeden raz. Samochód nie może mieć dwóch popsutych obiektów klasy Silnik czy jakiejkolwiek klasy dziedziczącej po klasie Czesc.
e) Zwróć obiekt klasy Samochod, poprawnie zainicjalizowany.

---

12. Możemy nieco skomplikować nasz algorytm tworzenia samochód, ale nie jest to obowiązkowe. To dobre ćwiczenie na wykorzystanie konstrukcji typu switch.

---

a) W metodzie stworzLosowySamochod() rozszerz algorytm losowania roku produkcji samochodu.
b) Przyjmij, że w zależności od wylosowanej marki samochodu może się zmieniać zakres lat produkcji dla samochodu. Przykładowo, dla marki BMW możemy losować rok produkcji pomiędzy 1990 i 2018, ale dla marki SKODA już między 2000 a 2018, dla FIAT np. między 2006 i 2018. Wybór jest dowolny, ale spróbuj zrobić tak, że przynajmniej 5 marek się od siebie różni, a dla 3 zakres jest ten sam.

---

13. Będziemy chcieli dowiedzieć się czegoś więcej o samochodach, więc przygotujemy do tego metodę toString oraz wyświetlimy kilka samochodów.

---

a) Analogicznie jak wykonałeś to w klasie Czesc i klasie Warsztat nadpisz metodę String toString() w klasie Samochod. Metoda powinna zwracać obiekt klasy String reprezentujący stan samochodu, a więc informacje o jego polach. W celu uzyskania reprezentacji w klasie String tablicy popsutych części możesz, jak wcześniej skorzystać z klasy Arrays.
b) Wróć teraz do klasy Warsztat i metody public static void main(String[] args).
c) Z użyciem metody stworzLosowySamochodZPopsutymSilniem utwórz 3 obiekty klasy Samochod i wyświetl je na konsoli. Upewnij się, że kolejne obiekty mają różne marki i różne lata produkcji oraz, że każdy z samochodów posiada pojedynczą popsutą część typu Silnik. Może się okazać, że twoja metoda toString w konkretnych klasach dziedziczących po klasie Czesc nie jest wystarczająco dobra. Jeżeli tak jest to popraw ją.
d) Z użyciem metody stworzLosowySamochod utwórz 10 obiektów klasy Samochod i wyświetl je na konsoli. Upewnij się, że kolejne obiekty mają różne marki, różne lata produkcji (jeżeli zastosowałeś algorytm z pkt. 12, to sprawdź czy odpowiednie marki otrzymują odpowiednie lata produkcji z zadanych przedziałów). Sprawdź również czy żaden samochód nie dostał mniej niż 1 popsutą część i nie więcej niż 5 popsutych części oraz czy każda z popsutych części jest innego typu.

CZĘŚĆ 3

Istnieje już nasz Warsztat oraz istnieją Samochody. Co więcej – znaleźliśmy sposób, aby wygodnie tworzyć losowe samochody, które będą symulować „klientów” przychodzących do warsztatu. Teraz pora wprawić wszystko w ruch i naprężyć programistyczne muskuły. Nie będzie z górki

14. Napiszmy główną pętlę naszej aplikacji, która będzie obsługiwać kolejnych klientów.

a) W metodzie main w klasie Warsztat stwórz nowy obiekt klasy Warsztat. Wykorzystaj statyczne metody dostarczające losowy zestaw części i obsługiwanych marek.

b) Stwórz nieskończoną pętle, w której będzie rozgrywało się życie naszego warsztatu.

c) Na początku zadbaj o to, aby wszystko nie działo się zbyt szybko, więc niech nasz program pośpi przez od 2 do 5 sekund (losowo). Będzie to symulowało pojawiających się klientów.

d) Stwórz losowy samochód, który pojawił się w Warsztacie. Możesz od razu go wyświetlić na konsoli – skorzystaj z metody toString() lub w bardziej elegancki sposób z użyciem klasy Formatter lub metody printf (System.out.printf(…))

15. W warsztacie jeszcze nie potrafimy naprawiać pojazdów, a już pierwszy się pojawił. Musimy uzbroić klasę Warsztat w odpowiednie narzędzia.

a) Stwórz metodę, która zwróci informację czy warsztat jest w stanie naprawić samochód. Sam dobierz nazwę metody i zwracaną wartość. Powinieneś wykorzystać informacje o obsługiwanych markach przez obiekt klasy Warsztat oraz o posiadanych częściach. Pamiętaj, że musisz skądś wiedzieć jakie części będą potrzebne. Skąd obiekt klasy Warsztat może to wiedzieć?

b) Stwórz metodę, która zwróci informację o koszcie naprawy wskazanego samochodu z uwzględnieniem kosztu potrzebnych części oraz marży warsztatu.

c) Stwórz metodę, która naprawi wskazany samochód, tzn. usunie z samochodu wszystkie popsute części oraz usunie z zestawu części w warsztacie te, który wykorzystano do naprawienia samochodu.

d) Zastanów się, co powinieneś zrobić gdyby w trakcie wywołania tych metod okazało się, że jest niewystarczająco części na warsztacie? Jak zabezpieczyć te metody przed taką sytuacją?

16. Warsztat potrafi już naprawiać samochody. Teraz niech to po prostu zrobi! … Aha! Jeszcze jedno – warsztat powinien na siebie zarabiać, jakoś gromadzić pieniądze i nie podpaść urzędowi skarbowemu.

a) Wróć do metody main w klasie Warsztat i głównej pętli aplikacji. Skoro w klasie Warsztat mamy już wszystkie metody, to trzeba po prostu je wykorzystać.

b) Dla wylosowanego w punkcie 14.d) samochodu sprawdź czy jest możliwy do obsłużenia przez Warsztat.

c) Jeżeli nie, bo marka się nie zgadza, to przejść do kolejnej iteracji.

d) Jeżeli tak, to przeprowadź naprawę i wystaw rachunek na odpowiednią kwotę. Ale czy pamiętasz o sytuacji, gdy marka jest „obsługiwana”, ale części brakuje? Jak to rozwiążesz? Czy napisane w punkcie 15. metody pozwalają Ci na to czy może musisz coś zmienić?

e) Zarobioną kwotę za naprawę trzeba gdzieś gromadzić. Zmodyfikuj klasę Warsztat tak, aby umożliwiała gromadzenie kasy.

f) Teraz jeszcze Urząd Skarbowy i mamy prawie wszystko. Samochód naprawiony, klient zapłacił, Warsztat się wzbogacił, ale należy się jeszcze paragon. Napisz prywatną metodę statyczną w klasie Warsztat, którą użyjesz w głównej pętli, aby wyświetlić elegancki paragon dla klienta.

g) Na sam koniec tej całej procedury – niech Warsztat pochwali się jeszcze stanem swojego konta. To też warto wyświetlić elegancko.

---

17. Wszystko już działa – Samochody się pojawiają, Warsztat je naprawia lub odrzuca, kasa się robi. No ale perpetuum mobile, to przecież jeszcze nie jest. Co jak części się w końcu skończą?

---

a) Zmodyfikuj metodę odpowiedzialną za naprawianie samochodów, tak, aby w jakiś sposób oznaczała, że zaczyna brakować części. Możesz przyjąć dowolny schemat postępowania. Np. gdy jakiejś części z danej klasy jest 0, to „alarmujemy” albo gdy potrzebnej części jest mniej niż 5, lub gdy wszystkich części jest mniej niż 20 albo gdy któregokolwiek z rodzaju części jest mniej niż 5. Wybór należy do Ciebie.

b) Napisz metodę, która dostarcza potrzebne części. Powinna ona w jakiś sposób zwiększać zasób części w Warsztacie. Jeżeli do tej pory pracowałeś na tablicy, to zastanów się co się stanie (w ogóle czy wybrany wariant dla punktu 17.a) to dopuszcza), jeżeli pojawi się więcej elementów niż początkowy maksymalny rozmiar tablicy? Potrafisz sobie poradzić z tą sytuacją?

c) Teraz należy stworzone mechanizmy wykorzystać. Przyjmij, że gdy części zabraknie, to zostaną one dostarczone po N-tym kliencie (samochodzie), gdzie N jest z zakresu do 3 do 10. Czy jeżeli pojawił się Samochód i zabrakło dla niego części (lub po jego naprawie zabrakło części), to kolejny po nim Samochód (klient) liczony jest jako 1, kolejny jako 2 itd. aż do N-tego.

---

18. Zapewniliśmy nieskończoną pracę naszego Warsztatu! Brawo! Możesz wszystko nieco skomplikować, którymś (albo wszystkimi) z podanych niżej sposobów.

---

a) Zmodyfikuj sposób obliczania czasu dostarczenia części w ten sposób, czas ten nie był losowy, a zależał od ilości i rodzaju potrzebnych części. Zmodyfikuj klasę Czesc i jej klasy Pochodne, aby informacja o czasie oczekiwania mogła być przez nie udostępniana i była inna w różnych klasach. Niech czas oczekiwania będzie sumą czasów oczekiwania na każdą pojedynczą część wyrażony w kolejnych iteracjach głównej pętli (kolejnych samochodach/klientach). Czyli jak

na części klasy Silnik czeka się 3 iteracje, a na Karoserie 2, to na dostawę 2 obiektów klasy Silnik i 3 obiektów klasy Karoseria będzie trzeba czekać 2*3 + 3*2 = 12 iteracji.

b) Obliczanie czasu oczekiwania z punktu powyżej zmień w ten sposób, aby czas potrzebny na dostarczenie części danego typu liczony (sumowany) był tylko raz. Czyli przyjmując czasy oczekiwania wyżej, to teraz na potrzebne części będziemy czekać 3 + 2 = 5 iteracji.

c) Wykorzystaj mechanizmy wielowątkowe, aby części pojawiły się asynchronicznie w naszym warsztacie. W tym celu usuń elementy odpowiedzialne za obsługę części z głównej pętli aplikacji. Stwórz nowy wątek, który po uruchomieniu będzie oczekiwał ilość sekund zgodną z ilością iteracji, a po zakończeniu tego czasu uzupełni zapasy części w warsztacie. Na koniec tego nowego wątku wyświetl na konsoli informację, że części zostały uzupełnione.

d) Wróć do miejsca gdzie tworzysz obiekt klasy Warsztat – jeszcze przed rozpoczęciem głównej pętli. Zanim utworzysz ten obiekt sprawdź czy istnieje w katalogu domowym plik „Warsztat.dat”. Jeżeli tak, to w tym pliku zapisany jest poprzedni stan warsztatu, przed zamknięciem aplikacji. Stwórz nowy obiekt Warsztatu na podstawie danych odczytanych z tego pliku (Sam ustal jaki będzie w pliku format do reprezentacji stanu Warsztatu).

e) Przed i po każdym nowym samochodzie do obsługi zapisz do pliku „Warsztat.dat” bieżący stan warsztatu, tak aby przy kolejnym uruchomieniu aplikacji dało się go odtworzyć. Format powinien być zgodny z tym z punktu d). Sam zadecyduj czy chcesz plik za każdym razem nadpisywać nowym stanem czy dopisywać do pliku nowy stan.

SKOŃCZYLIŚMY MOJE GRATULACJE!
