12. praktiskais darbs. 2. semestris

1. uzdevums

Izveidot vienu vai vairākas klases, kas neizmantojot Python iebūvēto datu struktūru kopa (set), bet izmanto kādu citu datu struktūru, piemēram, saraksts vai masīvs, realizē kopu izveidi un tipiskākās darbības ar tām (skat. lekcijas konspektu par kopām).

Kods:

```
# Programmas nosaukums: Klase "Kopa"

# 1. uzdevums (1MPR12_Vladislavs_Babaņins)

# Uzdevuma formulējums: Izveidot vienu vai vairākas klases, kas neizmantojot Python iebūvēto datu struktūru kopa (set),

# bet izmanto kādu citu datu struktūru, piemēram, saraksts vai masīvs, realizē kopu izveidi un tipiskākās darbības ar tām.

# Skat. lekcijas konspektu par kopām.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import copy

import random



class Kopa:

    # Klase kopa.

    def __init__(self):

        # Inicializēsim tukšo kopu.

        # self - kopa (objekts Kopa()).


        self.__kopa = []
```

```
    def saturs(self):
        # Atgriež kopas saturu.
        # self - kopa (objekts Kopa()).

        return self.__kopa

    def izdrukat(self):
        # Izdrukā glīta veidā kopas saturu.
        # Tukša kopa tiek attēlota ar Ø simbolu.
        # Piemēram, izdrūka { 1 2 3 }.
        # self - kopa (objekts Kopa()).

        sk = len(self.__kopa)
        if sk == 0:
            sv = "Ø"  # Tukša kopa Ø
        else:
            sv = "{ "
            for i in self.__kopa:
                sv = sv + str(i) + " "
            sv = sv + "}"
        print(sv)

    def pievienot(self, elements):
        # Ļauj pievienot vai nu vienu elementu kopai, vai nu elementus kā list (sarakstu), vai
tuple (kortežu).
        # Piemēram, a.pievienot([1, 3, 4]), vai b.pievienot((1, 2, 3)), vai c.pievienot(2).
        # self - kopa (objekts Kopa()).
        # elements - elements vai elementi, kurus gribam pievienot kopai.

        if type(elements) == list:
            for el in elements:
```

```
            if el not in self.__kopa:
                self.__kopa.append(el)

    elif type(elements) == tuple:
        for el in elements:
            if el not in self.__kopa:
                self.__kopa.append(el)
    else:
        paz = True
        for i in self.__kopa:
            if i == elements:
                paz = False
        if paz:
            self.__kopa.append(elements)

def izmest(self, elements):
    # Nodzēs norādītu vienu elementu (elements) no kopas.
    # Ja tāda elementa nav, tad neko nenodzēs.
    # Līdzīgs set discard.
    # self - kopa (objekts Kopa()).
    # elements - elements vai elementi, kurus gribam pievienot kopai.

    for i in self.__kopa:
        if i == elements:
            self.__kopa.remove(elements)

def nonemt(self, elements):
    # Nodzēs norādītu vienu elementu (elements) no kopas.
    # Ja tāda elementa nav, tad raise ValueError, ka tāds elements nav kopā.
    # Līdzīgs set remove.
    # self - kopa (objekts Kopa()).
```

```
        # elements - elements vai elementi, kurus gribam pievienot kopai.

        if elements not in self.__kopa:
            raise ValueError(f"{elements} nav kopā. Nevar noņemt elementu, kuru nav. Izmantojiet 'izmest'.")  # is not in the kopa

        for i in self.__kopa:
            if i == elements:
                self.__kopa.remove(elements)

    def vai_pieder(self, elements):
        # Pārbauda vai kopai piedēr norādītais elements.
        # Atgriež True, ja norādītais elements piedēr kopai.
        # Atgriež False, ja norādītais elements nepiedēr kopai.
        # self - kopa (objekts Kopa()).
        # elements - elements vai elementi, kurus gribam pievienot kopai.

        for i in self.__kopa:
            if i == elements:
                return True
        return False

    def izdzest(self):
        # Pārverš kopu par tukšo kopu un atgriež to.
        # self - kopa (objekts Kopa()).

        self.__kopa = []
        return self.__kopa

    def atjaunot(self, elements):
        # Vai pārmainīt visas vērtības kopai.
```

```
        # Piemēram, a.update([4, 5, 6, 7]).
        # Atjauno kopu līdzīgi, ka set update.
        # self - kopa (objekts Kopa()).
        # elements - elements vai elementi, kurus gribam pievienot kopai.

        self.__kopa = []

        if type(elements) == list:
            for el in elements:
                if el not in self.__kopa:
                    self.__kopa.append(el)

        elif type(elements) == tuple:
            for el in elements:
                if el not in self.__kopa:
                    self.__kopa.append(el)
        else:
            paz = True
            for i in self.__kopa:
                if i == elements:
                    paz = False
            if paz:
                self.__kopa.append(elements)

    def pop(self):
        # Atgriež vienu nejaušu elementu no kopas.
        # Paņem nejaušu kopas elementu līdzīgi, ka set pop.
        # Ja mēģināsim izņemt nejaušu elementu no tukšas kopas, tad print("Kļūda! Nevar
izņemt elementu no tukšas kopas.").
        # self - kopa (objekts Kopa()).
```

```
    try:
        a = random.choice(self.__kopa)
    except:
        print("Kļūda! Nevar izņemt elementu no tukšas kopas.")  # Raise Exception
    else:
        self.__kopa.remove(a)
        return a

@staticmethod
def apvienot(a, b):
    # Apvieno divas kopas a un b.
    # A U B
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).

    c = copy.deepcopy(a)
    x = a.saturs()
    y = b.saturs()
    for i in y:
        if i not in x:
            c.pievienot(i)
    return c

# Var izmantot arī šo metodi, "apvienot1" nevis "apvienot".
# Metode "apvienot1" neizmanto deepcopy.
'''
def apvienot1(a, b): # tas pats, bet bez deepcopy
    c = Kopa()
    x = a.saturs()
    y = b.saturs()
    for z in x:
```

```
            c.pievienot(z)
        for z in y:
            if z not in x:
                c.pievienot(z)

        return c
    '''

    @staticmethod
    def vai_parklajas(a, b):
        # Pārbauda vai divas kopas a un b pārklājas.
        # Atgriež True, ja kopas pārklājas (ir vismaz viens kopīgs elements).
        # Atgriež False, ja kopas nepārklājas (nav nevienā kopīga elementa).
        # a - 1.kopa (objekts Kopa()).
        # b - 2.kopa (objekts Kopa()).

        c = Kopa()
        x = a.saturs()
        y = b.saturs()
        tuksa_kopa = Kopa()

        d = c.skelums(a, b)
        if Kopa.vai_vienadas(d, tuksa_kopa):
            return False
        return True

    @staticmethod
    def vai_neparklajas(a, b):
        # Pārbauda vai divas kopas a un b nepārklājas (tas pats kā vai_parklajas, bet tikai
negācija).
        # Atgriež False, ja kopas pārklājas (ir vismaz viens kopīgs elements).
```

```
    # Atgriež True, ja kopas nepārklājas (nav nevienā kopīga elementa).
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).

    c = Kopa()
    x = a.saturs()
    y = b.saturs()
    tuksa_kopa = Kopa()

    d = c.skelums(a, b)
    if Kopa.vai_vienadas(d, tuksa_kopa):
        return True
    return False

@staticmethod
def skelums(a, b):
    # Atgriež kopas a un b šķēlumu (A ∩ B).
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).

    c = Kopa()
    x = a.saturs()
    y = b.saturs()
    for i in x:
        if i in y:
            c.pievienot(i)
    return c

@staticmethod
def simetriska_starpiba(a, b):
    # Atgriež kopas a un b simetrisku starpību (A △ B).
```

```
    # Pēc formulas: A △ B = (A \ B) U (B \ A).
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).

    c = Kopa()
    d = Kopa.starpiba(a, b)
    f = Kopa.starpiba(b, a)
    c = Kopa.apvienot(d, f)
    return c

@staticmethod
def starpiba(a, b):
    # Atgriež kopas a un b starpību (A \ B).
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).

    c = Kopa()
    x = a.saturs()
    y = b.saturs()
    for i in x:
        if i not in y:
            c.pievienot(i)
    return c

@staticmethod
def vai_vienadas(a, b):
    # Pārbauda vai divas kopas a un b ir vienādas (A == B).
    # Atgriež True, ja divas kopas ir vienādas.
    # Atgriež False, ja divas kopas nav vienādas.
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).
```

```
    x = a.saturs()
    y = b.saturs()
    for i in x:
        if i not in y:
            return False

    for i in y:
        if i not in x:
            return False

    return True

@staticmethod
def vai_nav_vienadas(a, b):
    # Pārbauda vai divas kopas a un b nav vienādas (A != B).
    # Atgriež True, ja divas kopas nav vienādas.
    # Atgriež False, ja divas kopas ir vienādas.
    # a - 1.kopa (objekts Kopa()).
    # b - 2.kopa (objekts Kopa()).

    x = a.saturs()
    y = b.saturs()
    for i in x:
        if i not in y:
            return True

    for i in x:
        if i not in y:
            return True
```

```
        return False


    @staticmethod
    def vai_apakskopa(a, b):
        # Pārbauda vai a ir apakškopa b. A ⊆ B.
        # Atgriež True, ja a ir apakškopa b.
        # Atgriež False, ja a nav apakškopa b.
        # a - 1.kopa (objekts Kopa()).
        # b - 2.kopa (objekts Kopa()).

        x = a.saturs()
        y = b.saturs()
        for i in x:
            if i not in y:
                return False
        return True


    @staticmethod
    def vai_stingra_apakskopa(a, b):
        # Pārbauda vai a ir stingra apakškopa b. A ⊂ B. (ja A un B ir vienādas kopas, tad A nav
stingra apakškopa B).
        # Atgriež True, ja a ir stingra apakškopa b.
        # Atgriež False, ja a nav stingra apakškopa b.
        # a - 1.kopa (objekts Kopa()).
        # b - 2.kopa (objekts Kopa()).

        x = a.saturs()
        y = b.saturs()
        for i in x:
            if i not in y:
                return False
```

```
        if Kopa.vai_vienadas(a, b):
            print(Kopa.vai_vienadas(a, b))
            return False
        else:
            return True



# ------------------------------------------------------
# Galvenā programmas daļa
# ------------------------------------------------------



# Pārbauda klases metodes.
kopa1 = Kopa()
kopa1.pievienot(5)
kopa1.pievienot(6)
kopa1.pievienot(3)
kopa1.pievienot(3)  # Pārbaudam vai pievienos vienādus elementus pa vienam.
print("1.kopa:")
kopa1.izdrukat()
print()

kopa2 = Kopa()
kopa2.pievienot([1, 2, 3, 3])  # Pārbaudam vai pievienos vienādus elementus sarakstā (nepievienos).
print("2.kopa:")
kopa2.izdrukat()
print()

kopa3 = Kopa()
```

```
kopa3.pievienot((10, 11))  # Pārbaudam vai pievienos vienādus elementus sarakstā (nepievienos).

print("3.kopa:")

kopa3.izdrukat()

print()


kopa4 = Kopa()

kopa4.pievienot(15)

kopa4.pievienot(16)

print("4.kopa:")

kopa4.izdrukat()

print()


print("Izdzēš 3.kopu (pārverš par tukšu kopu).")

kopa3.izdzest()  # Tāgad pārvēršam par tukšu kopu.

print("3.kopa:")

kopa3.izdrukat()

print()


pop1_no_kopas_4 = kopa4.pop()  # ar izveidoto pop funkciju paņēmam vienu nejaušu elementu no kopas un tas tiek nodzēsts sākotnēja kopā. (tiek definēta klasē neizmantojot set pop).

print("Pirmais nejauši paņemts elements no 4.kopas (un tas tika izņemts ara):")

print(pop1_no_kopas_4)

print()


print("4.kopa, kad mēs nejauši izņēmam nejauši elementu '" + str(pop1_no_kopas_4) + "' :")

kopa4.izdrukat()

print()


pop2_no_kopas_4 = kopa4.pop()  # ar izveidoto pop funkciju paņēmam vel vienu nejaušu elementu no kopas un tas tiek nodzēsts sākotnēja kopā.
```

```
print("Otrais nejauši paņemts elements no 4.kopas (un tas tika izņemts ara):")
print(pop2_no_kopas_4)
print()

print("4.kopa, kad mēs nejauši izņēmam elementu '" + str(pop2_no_kopas_4) + "' :")
kopa4.izdrukat()
print()

print("4.kopa, kad mēs nejauši izņēmam vel vienu elementu:")
pop3_no_kopas_4 = kopa4.pop()
print()

print("4.kopa:")
kopa4.izdrukat()
print()

print("Izveidojam jaunu 5.kopu:")
kopa5 = Kopa()
kopa5.izdrukat()
print()

print("Izveidojam jaunu 6.kopu:")
kopa6 = Kopa()
kopa6.pievienot(1)
kopa6.pievienot(2)
kopa6.pievienot(3)
kopa6.pievienot(3)
kopa6.izdrukat()
print()

kopa5.pievienot([1, 2, 4])
```

```
print("Pievienojam 5.kopai elementus 1, 2, 4:")

kopa5.izdrukat()

print()


kopa5.pievienot([1, 3, 4])

print("Pievienojam 5.kopai elementus 1, 3, 4:")

kopa5.izdrukat()

print()


kopa5.atjaunot((10, 12, 13))

print("Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķīram jaunus elementus 10, 12, 13:")

kopa5.izdrukat()

print()


kopa5.atjaunot(10)

print("Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķīram jaunu elementu 10:")

kopa5.izdrukat()

print()


kopa5.atjaunot([])

print("Atjaunojam 5.kopu, nodzēsam visus elementus kopā:")

kopa5.izdrukat()

print()


kopa5.atjaunot([2, 3, 4])

print("Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķīram jaunus elementus 2, 3, 4:")

kopa5.izdrukat()

print()
```

```
print("5.kopa:")
kopa5.izdrukat()
print()

print("6.kopa:")
kopa6.izdrukat()
print()

a = Kopa.apvienot(kopa5, kopa6)
print("5.kopas apvienojums ar 6.kopu:")
a.izdrukat()
print()

a = Kopa.skelums(kopa5, kopa6)
print("5.kopas šķēlums ar 6.kopu:")
a.izdrukat()
print()

a = Kopa.simetriska_starpiba(kopa5, kopa6)
print("5.kopas simetriska starpība ar 6.kopu:")
a.izdrukat()
print()

a = Kopa.starpiba(kopa5, kopa6)
print("5.kopas starpība ar 6.kopu:")
a.izdrukat()
print()

a = Kopa.vai_parklajas(kopa5, kopa6)
print("Vai 5.kopa pārklajas ar 6.kopu:")
print(a)  # uzrakstīs True vai False
```

```
print()

a = Kopa.vai_neparklajas(kopa5, kopa6)
print("Vai 5.kopa nepārklajas ar 6.kopu:")
print(a)  # uzrakstīs True vai False
print()

a = Kopa.vai_vienadas(kopa5, kopa6)
print("Vai 5.kopa ir vienāda ar 6.kopu:")
print(a)  # uzrakstīs True vai False
print()

a = Kopa.vai_nav_vienadas(kopa5, kopa6)
print("Vai 5.kopa nav vienāda ar 6.kopu:")
print(a)  # uzrakstīs True vai False
print()

a = Kopa.vai_stingra_apakskopa(kopa5, kopa6)
print("Vai 5.kopa ir stingra apakškopa 6.kopai:")
print(a)
print()

a = Kopa.vai_apakskopa(kopa5, kopa6)
print("Vai 5.kopa ir 6.kopas apakškopa:")
print(a)  # uzrakstīs True vai False
print()

n = 2
a = kopa5.vai_pieder(n)
print("Vai 5.kopai")
kopa5.izdrukat()
```

```
print("pieder elements " + str(n) + ":")
print(a)  # uzrakstīs True vai False
print()

n = 6
a = kopa6.vai_pieder(n)
print("Vai 6.kopai")
kopa6.izdrukat()
print("pieder elements " + str(n) + ":")
print(a)  # uzrakstīs True vai False
print()

kopa7 = Kopa()
kopa7.pievienot(1)
kopa7.pievienot(2)
kopa7.pievienot(3)

print("7.kopa:")
kopa7.izdrukat()
print()

print("Izmest '1' no 7.kopas.")
kopa7.izmest(1)
print("7.kopa:")
kopa7.izdrukat()
print()

print("Izmest '14' no 7.kopas.")
kopa7.izmest(14)
print("7.kopa:")
kopa7.izdrukat()
```

```
print()

print("Noņemt '2' no 7.kopas.")
kopa7.nonemt(2)
print("7.kopa:")
kopa7.izdrukat()
print()

print("Noņemt '3' no 7.kopas.")
kopa7.nonemt(3)
print("7.kopa:")
kopa7.izdrukat()
print()

kopa8 = Kopa()
kopa8.pievienot("a")
kopa8.pievienot("b")
kopa8.pievienot(["c", "d", 5])
print("8.kopa:")
kopa8.izdrukat()
```

Testa piemēri:

1.kopa:

{ 5 6 3 }

2.kopa:

{ 1 2 3 }

3.kopa:

{ 10 11 }

4.kopa:

{ 15 16 }

Izdzēš 3.kopu (pārverš par tukšu kopu).

3.kopa:

Ø

Pirmais nejauši paņemts elements no 4.kopas (un tas tika izņemts ara):

16

4.kopa, kad mēs nejauši izņēmam nejauši elementu '16' :

{ 15 }

Otrais nejauši paņemts elements no 4.kopas (un tas tika izņemts ara):

15

4.kopa, kad mēs nejauši izņēmam elementu '15' :

Ø

4.kopa, kad mēs nejauši izņēmam vel vienu elementu:

Kļūda! Nevar izņemt elementu no tukšas kopas.

4.kopa:

Ø

Izveidojam jaunu 5.kopu:

Ø

Izveidojam jaunu 6.kopu:

{ 1 2 3 }

Pievienojam 5.kopai elementus 1, 2, 4:

{ 1 2 4 }

Pievienojam 5.kopai elementus 1, 3, 4:

{ 1 2 4 3 }

Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķīram jaunus elementus 10, 12, 13:

{ 10 12 13 }

Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķīram jaunu elementu 10:

{ 10 }

Atjaunojam 5.kopu, nodzēsam visus elementus kopā:

Ø

Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķīram jaunus elementus 2, 3, 4:

{ 2 3 4 }

5.kopa:

{ 2 3 4 }

6.kopa:

{ 1 2 3 }

5.kopas apvienojums ar 6.kopu:

{ 2 3 4 1 }

5.kopas šķēlums ar 6.kopu:

{ 2 3 }

5.kopas simetriska starpība ar 6.kopu:

{ 4 1 }

5.kopas starpība ar 6.kopu:

{ 4 }

Vai 5.kopa pārklajas ar 6.kopu:

True

Vai 5.kopa nepārklajas ar 6.kopu:

False

Vai 5.kopa ir vienāda ar 6.kopu:

False

Vai 5.kopa nav vienāda ar 6.kopu:

True

Vai 5.kopa ir stingra apakškopa 6.kopai:

False

Vai 5.kopa ir 6.kopas apakškopa:

False

Vai 5.kopai

{ 2 3 4 }

pieder elements 2:

True

Vai 6.kopai

{ 1 2 3 }

pieder elements 6:

False

7.kopa:

{ 1 2 3 }

Izmest '1' no 7.kopas.

7.kopa:

{ 2 3 }

Izmest '14' no 7.kopas.

7.kopa:

{ 2 3 }

Noņemt '2' no 7.kopas.

7.kopa:

{ 3 }

Noņemt '3' no 7.kopas.

7.kopa:

Ø

8.kopa:

{ a b c d 5 }

2)

```
1.kopa:
{ 5 6 3 }

2.kopa:
{ 1 2 3 }

3.kopa:
{ 10 11 }

4.kopa:
{ 15 16 }

Izdzēš 3.kopu (pārverš par tukšu kopu).
3.kopa:
∅

Pirmais nejauši paņemts elements no 4.kopas (un tas tika izņemts ara):
16

4.kopa, kad mēs nejauši izņēmam nejauši elementu '16' :
{ 15 }

Otrais nejauši paņemts elements no 4.kopas (un tas tika izņemts ara):
15

4.kopa, kad mēs nejauši izņēmam elementu '15' :
∅

4.kopa, kad mēs nejauši izņēmam vel vienu elementu:
Kļūda! Nevar izņemt elementu no tukšas kopas.

4.kopa:
∅

Izveidojam jaunu 5.kopu:
∅

Izveidojam jaunu 6.kopu:
{ 1 2 3 }

Pievienojam 5.kopai elementus 1, 2, 4:
{ 1 2 4 }

Pievienojam 5.kopai elementus 1, 3, 4:
{ 1 2 4 3 }
```

3)

```
Atjaunojam 5.kopu, nodzēsam visus elementus kopā un piešķiram jaunus elementus 2, 3, 4:
{ 2 3 4 }

5.kopa:
{ 2 3 4 }

6.kopa:
{ 1 2 3 }

5.kopas apvienojums ar 6.kopu:
{ 2 3 4 1 }

5.kopas šķēlums ar 6.kopu:
{ 2 3 }

5.kopas simetriska starpība ar 6.kopu:
{ 4 1 }

5.kopas starpība ar 6.kopu:
{ 4 }

Vai 5.kopa pārklajas ar 6.kopu:
True

Vai 5.kopa nepārklajas ar 6.kopu:
False

Vai 5.kopa ir vienāda ar 6.kopu:
False

Vai 5.kopa nav vienāda ar 6.kopu:
True

Vai 5.kopa ir stingra apakškopa 6.kopai:
False

Vai 5.kopa ir 6.kopas apakškopa:
False

Vai 5.kopai
{ 2 3 4 }
pieder elements 2:
True
```

2. uzdevums

Sastādīt programmu, kas realizē vienkāršotu automašīnu nomas punkta darbību. Nomas punktā ir vairāku tipu automašīnas - vieglās automašīnas, kravas automašīnas un autobusi. Auto nomas punkts var iegādāties jaunas automašīnas un utilizēt nevajadzīgās automašīnas. Auto nomas punkts prot paziņot, kādas automašīnas tas pašlaik piedāvā. Auto nomas punkta klienti ir juridiskas un fiziskas personas. Auto nomas klients var veikts divas darbības - nomāt tikai Auto nomas punktā esošu automašīnu vai atdot atpakaļ nomāto automašīnu.

Kods: (ar vadītāju kategorijām un auto ID)

```
# Programmas nosaukums: Automašīnu nomas simulācija.

# 2. uzdevums (1MPR12_Vladislavs_Babaņins).

# Uzdevuma formulējums: Sastādīt programmu, kas realizē vienkāršotu automašīnu nomas punkta darbību.

# Nomas punktā ir vairāku tipu automašīnas - vieglās automašīnas, kravas automašīnas un autobusi.

# Auto nomas punkts var iegādāties jaunas automašīnas un utilizēt nevajadzīgās automašīnas.

# Auto nomas punkts prot paziņot, kādas automašīnas tas pašlaik piedāvā.

# Auto nomas punkta klienti ir juridiskas un fiziskas personas. Auto nomas klients var veikts divas darbības - nomāt tikai Auto nomas punktā esošu automašīnu vai atdot atpakaļ nomāto automašīnu.

# Programmas autors: Vladislavs Babaņins.

# Versija 1.0


"""

Ar vadītāju kategorijām un auto ID.

"""




class Buyer:

    # Klase pircējam.

    def __init__(self, name, license, is_individual):

        # name - vārds un uzvārds vai kompānija.

        # license - vadītāju apliecības a. "A" vai "B" vai "C" vai "D" vai "nav".
```

```
        # is_individual - True - ir fiziska persona. False - juridiska persona.

        # A - vadītāju apliecības kategorija motocikliem (šajā programmā nav motociklus, bet tā ir reāla kategorija, tāpēc to ierākstīju).

        # B - vadītāju apliecības kategorija viegliem automobiļiem.

        # C - vadītāju apliecības kategorija kravas automobiļiem.

        # D - vadītāju apliecības kategorija autobusiem automobiļiem.

        self.name = name

        self.license = license

        self.is_individual = is_individual


        # Pārbauda vai licence nav A vai B vai C vai D vai nav.

        # Ja nav, tad license - nav.

        if self.license != "A" and self.license != "B" and self.license != "C" and self.license != "D" and self.license != "nav":

            if self.is_individual:  # Fiziskām personam.

                print(f"Pievienota fiziskā persona {self.name} bez vadītāja apliecību.")  # Ja nesakrīt ne ar vienu reālu apliecības kategoriju, tad nav vadītaja apliecības fiziskai personai.

            elif not self.is_individual:  # Juridiskām personam.

                print(f"Pievienota juridiskā persona {self.name} bez vadītāja apliecību.")  # Ja nesakrīt ne ar vienu reālu apliecības kategoriju, tad nav vadītaja apliecības juridiskai personai.

            else:  # Kļūda ja netika ierākstīta Boolean vērtība (True vai False).

                print(f"Kļūda! {self.name} personai jābut vai nu fiziskai vai juridiskai!")

            self.license = "nav"


        elif self.license == "A":  # Ja ir A kategorija (motocikliem) apliecībai.

            if self.is_individual:  # Ja ir A kategorija apliecībai un tā ir fiziska persona.

                print(f"Pievienota fiziskā persona {self.name} ar A kategorijas (motocikli) vadītāja apliecību.")

            else:   # Ja ir A kategorija apliecībai un tā ir juridiskā persona.

                print(f"Pievienota juridiskā persona {self.name} ar A kategorijas (motocikli) vadītāja apliecību.")
```

```
        elif self.license == "B":  # Ja ir B (vieglauto) kategorija apliecībai.
            if self.is_individual:  # Ja ir B kategorija apliecībai un tā ir fiziska persona.
                print(f"Pievienota fiziskā persona {self.name} ar B kategorijas (vieglauto) vadītāja apliecību.")
            else:  # Ja ir B kategorija apliecībai un tā ir juridiskā persona.
                print(f"Pievienota juridiskā persona {self.name} ar B kategorijas (vieglauto) vadītāja apliecību.")

        elif self.license == "C":  # Ja ir C (kravas automobiļiem) kategorija apliecībai.
            if self.is_individual:  # Ja ir C kategorija apliecībai un tā ir fiziska persona.
                print(f"Pievienota fiziskā persona {self.name} ar C kategorijas (kravas automašīnas) vadītāja apliecību.")
            else:  # Ja ir C kategorija apliecībai un tā ir juridiskā persona.
                print(f"Pievienota juridiskā persona {self.name} ar C kategorijas (kravas automašīnas) vadītāja apliecību.")

        elif self.license == "D":  # Ja ir D (autobusiem) kategorija apliecībai.
            if self.is_individual:  # Ja ir D kategorija apliecībai un tā ir fiziska persona.
                print(f"Pievienota fiziskā persona {self.name} ar D kategorijas (autobuss) vadītāja apliecību.")
            else:  # Ja ir D kategorija apliecībai un tā ir juridiskā persona.
                print(f"Pievienota juridiskā persona {self.name} ar D kategorijas (autobuss) vadītāja apliecību.")

    def print_license(self):
        # Izvadīt vadītāju apliecības kategoriju uz ekrāna.
        if self.license == "nav":  # Ja nav vadītāju apliecības kategoriju cilvēkam, tad izvadam kā nav.
            print(f"{self.name} nav vadītāja apliecības.")
        else:  # Ja vadītāju apliecības kategorija ir cilvēkam, tad izvadam ka ta ir.
            print(f"{self.name} ir vadītāja apliecība kategorijā {self.license}.")

    def return_license(self):
```

```
        # Atgriež vadītāju apliecības kategoriju uz ekrāna.
        return self.license

    def return_name(self):
        # Atgriež pircēja vārdu vai uzvārdu.
        return self.name


class Car:
    id_counter = 0  # Lai sekotu līdzi ID numuram.

    def __init__(self, model):
        # model - nosaukums automašīnai.
        self.id = Car.id_counter  # piešķirt automašīnai unikālu ID.
        Car.id_counter += 1  # ID skaitītājs pieaug par vienu nākamajai automašīnai.
        self.model = model
        self.is_rented = False

    def return_name(self):
        # Atgriež automašīnas nosaukumu.
        return self.model


class Truck(Car):
    # Apakšklase Car klasei.
    # Šai klasei ir cargo_capacity (kravas kapacitāte, vai kravas automašīnas ietilpība).
    def __init__(self, model, cargo_capacity):
        # model - automašīnas nosaukums.
        # cargo_capacity - kravas kapacitāte.
        super().__init__(model)
        self.cargo_capacity = cargo_capacity
```

```
    def return_name(self):
        # Atgriež kravas automašīnas nosaukumu.
        return self.model


class Bus(Car):
    def __init__(self, model, num_of_seats):
        # model - automašīnas nosaukums.
        # cargo_capacity - vietu skaits automašīnā.
        super().__init__(model)
        self.num_of_seats = num_of_seats

    def return_name(self):
        # Atgriež autobusa nosaukumu.
        return self.model


class Rental_point:
    # Nomas punkta klase.
    def __init__(self):
        self.cars = []  # Saraksts, kurā glabāsim visas automašīnas, kuri ir pieejāmi nomašanai.
        self.buses = []  # Saraksts, kurā glabāsim visus autobusus, kuri ir pieejāmi nomašanai.
        self.trucks = []  # Saraksts, kurā glabāsim visus kravas automašīnas, kuri ir pieejāmi
nomašanai.
        self.rented_vehicles = []  # Lai sekotu līdzi iznomātajiem transportlīdzekļiem. Saraksts ar
transportlīdzekļiem, kuri nav pieejāmi.

    def add_car(self, car):
        # car - objekts no Car klases.
        print(f"Automašīna {car.model} tika nogādāta nomas punktā!")
        self.cars.append(car)
```

```
    def add_bus(self, bus):
        # bus - objekts no Bus klases (apakšklase Car'am).
        print(f"Autobuss {bus.model} tika nogādāts nomas punktā!")
        self.buses.append(bus)

    def add_truck(self, truck):
        # truck - objekts no Truck klases (apakšklase Car'am).
        print(f"Kravas automašīna {truck.model} tika nogādāta nomas punktā!")
        self.trucks.append(truck)

    def remove_car(self, car):
        # Nodzēst vienu noteiktu vieglauto no saraksta ar pieejāmam automašīnam (nodot vieglauto metāllūžņos).
        if car in self.cars:  # Ja tas vieglauto ir sarakstā ar pieejāmam automašīnam.
            print(f"Nomas punkts nodeva {car.model} metāllūžņos!")  # Izvadīt informāciju par to, kādu vieglauto nodzēsam.
            self.cars.remove(car)  # Nodzēst no saraksta ar pieejāmiem automašīnam šo konkrētu vieglauto.
        else:  # Nevar nodot metallužnos vieglauto, ja tādas nav.
            print(f"Nevar nodot metallužnos automašīnu {car.model}, jo tādas automašīnas nav nomas punktā!")  # Izvadīt informāciju par to, ka nevaram nodzēst vieglauto, kura nav pieejāma.

    def remove_bus(self, bus):
        # Nodzēst vienu noteiktu autobusu no saraksta ar pieejāmam automašīnam (nodot autobusu metāllūžņos).
        if bus in self.buses:  # Ja tas autobuss ir sarakstā ar pieejāmam automašīnam.
            print(f"Nomas punkts nodeva autobusu {bus.model} metāllūžņos!")  # Izvadīt informāciju par to, kādu autobusu nodzēsam.
            self.buses.remove(bus)  # Nodzēst no saraksta ar pieejāmiem automašīnam šo konkrētu autobusu.
        else:  # Nevar nodot metallužnos autobusu, ja tādu nav.
```

```
        print(f"Nevar nodot metallužnos autobusu {bus.model}, jo tāda autobusa nav nomas punktā!")  # Izvadīt informāciju par to, ka nevaram nodzēst autobusu, kurš nav pieejāms.


    def remove_truck(self, truck):
        # Nodzēst vienu noteiktu kravas automašīnu no saraksta ar pieejāmam automašīnam (nodot kravas automašīnu metāllūžņos).
        if truck in self.trucks:  # Ja tas kravas automašīna ir sarakstā ar pieejāmam automašīnam.
            print(f"Nomas punkts nodeva kravas automašīnu {truck.model} metāllūžņos!")  # Izvadīt informāciju par to, kādu kravas automašīnu nodzēsam.
            self.trucks.remove(truck)  # Nodzēst no saraksta kravas automašīnu.
        else:  # Nevar nodot metallužnos kravas automašīnu, ja tādas nav.
            print(f"Nevar nodot metallužnos kravas automašīnu {truck.model}, jo tādas kravas automašīnas nav nomas punktā!")  # Izvadīt informāciju par to, ka nevaram nodzēst kravas automašīnu, kura nav pieejāma.


    def announce_cars(self):
        # Izvadīt (iz-print'ēt) uz ekrāna automašīnas.
        print("Pašlaik nomai pieejami vieglauto:")

        free_auto = []
        for car in self.cars:
            if not car.is_rented:
                free_auto.append(car.model)

        if free_auto:
            for model in free_auto:
                print(model)
        else:
            print("Nav brīvu vieglauto.")
        print()


    def announce_buses(self):
```

```
    # Izvadīt (iz-print'ēt) uz ekrāna autobusus.
    print("Pašlaik nomai pieejami autobusi:")

    free_buses = []
    for bus in self.buses:
        if not bus.is_rented:
            free_buses.append(bus.model)

    if free_buses:
        for model in free_buses:
            print(model)
    else:
        print("Nav brīvu autobusu.")
    print()

def announce_trucks(self):
    # Izvadīt (iz-print'ēt) uz ekrāna kravas automašīnas.
    print("Pašlaik nomai pieejami kravas auto:")

    free_trucks = []
    for truck in self.trucks:
        if not truck.is_rented:
            free_trucks.append(truck.model)

    if free_trucks:
        for model in free_trucks:
            print(model)
    else:
        print("Nav brīvu kravas auto.")

    print()
```

```
    def announce_available_vehicles(self):
        # Izvadīt (iz-print'ēt) uz ekrāna automašīnas.
        # Izvadīt (iz-print'ēt) uz ekrāna autobusus.
        # Izvadīt (iz-print'ēt) uz ekrāna kravas automašīnas.
        self.announce_cars()
        self.announce_buses()
        self.announce_trucks()

    def sell_car(self, item, buyer):
        # Automašīnas nodošanai nomā.
        if isinstance(buyer, Buyer):
            if isinstance(item, Car) and not item.is_rented and item in self.cars:
                if buyer.license == "B":
                    item.is_rented = True
                    self.rented_vehicles.append(item)
                    print(f"{item.model} tika nodots nomai! Tagad to noma {buyer.name}.")
                else:
                    print(f"{buyer.name} ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu
vieglauto {item.model}.")

            elif isinstance(item, Bus) and not item.is_rented and item in self.buses:
                if buyer.license == "D":
                    item.is_rented = True
                    self.rented_vehicles.append(item)
                    print(f"{item.model} tika nodots nomai! Tagad to noma {buyer.name}.")
                else:
                    print(f"{buyer.name} ir nepieciešamas D kategorijas vadītāja apliecība, lai nomātu
autobusu {item.model}.")

            elif isinstance(item, Truck) and not item.is_rented and item in self.trucks:
                if buyer.license == "C":
```

```
                item.is_rented = True
                self.rented_vehicles.append(item)
                print(f"{item.model} tika nodots nomai! Tagad to noma {buyer.name}." +
str(buyer.return_license()))
            else:
                print(f"{buyer.name} ir nepieciešamas C kategorijas vadītāja apliecība, lai nomātu
kravas automašīnu {item.model}." + str(buyer.return_license()))
        else:
            if isinstance(item, Bus):
                print(f"Atvainojiet, {buyer.name}! {item.model} pašlaik nav pieejams autobusu
nomas punktā!")
            else:
                print(f"Atvainojiet, {buyer.name}! {item.model} pašlaik nav pieejama
automašīnas nomas punktā!")
    else:
        print(f"Kļūda! {buyer.name} ir nepieciešama noteikta vadītāja apliecības kategorija, lai
nomātu {item.model}!")

  def return_vehicle(self, vehicle):
    # Atgriezt transportlīdzekļu automašīnas nomai.
    if vehicle in self.rented_vehicles:
      vehicle.is_rented = False
      self.rented_vehicles.remove(vehicle)
      print(f"{vehicle.model} tika atgriezts nomas punktā.")
    else:
      print(f"{vehicle.model} nebija nomāts šajā nomas punktā!")


# TESTA PIEMĒRI.
# Izveidot Pircēja klases instances.
buyer1 = Buyer("Artūrs Kariņš", "A", True)
buyer2 = Buyer("SIA 'RVR'", "B", False)
buyer3 = Buyer("Edgars Pliekšāns", "E", True)
```

```
buyer4 = Buyer("AS 'Gosbank'", "D", False)
buyer5 = Buyer("Jāzeps Baumanis", "nav", False)
buyer6 = Buyer("SIA 'OGAS'", "F", False)
buyer7 = Buyer("Lizete Rozenberga", "C", True)
buyer8 = Buyer("SIA 'OBKhSS'", "B", False)

# Auto klases instances izveide.
volga = Car("GAZ-24 Volga")
zhiguli = Car("VAZ-2106 Zhiguli")
moskvich = Car("Moskvich-403")
pioneer = Car("Pioneer 2M")
lamborghini = Car("Lamborghini Diablo")
rolls_royce = Car("Rolls-Royce Phantom")

# Truck klases instances izveide.
hummer = Truck("Hummer H3", 5000)
hanomag = Truck("Hanomag-Henschel F55", 10000)
uaz = Truck("UAZ-452", 1500)

# Autobusu klases instances izveide.
mercedes = Bus("Mercedes-Benz Sprinter", 12)
raf = Bus("RAF-2203", 15)

# Rental_point klases instances izveide.
rental_point = Rental_point()
print()

# Pievienojam automašīnas nomas punktam.
rental_point.add_car(volga)
rental_point.add_car(zhiguli)
rental_point.add_car(moskvich)
```

```
rental_point.add_car(pioneer)
rental_point.add_car(lamborghini)
rental_point.add_car(rolls_royce)
print()

# Pārbaude kravas automašīnu pievienošanu nomas punktam.
rental_point.add_truck(hummer)
rental_point.add_truck(hanomag)
rental_point.add_truck(uaz)
print()

# Testē autobusu pievienošanu nomas punktam.
rental_point.add_bus(mercedes)
rental_point.add_bus(raf)
print()

rental_point.announce_available_vehicles()

# Pārbaude automašīnu izņemšanu no nomas punkta.
rental_point.remove_car(lamborghini)
rental_point.remove_car(rolls_royce)

# Kravas automašīnu izņemšanas pārbaude no nomas punkta.
rental_point.remove_truck(hummer)
rental_point.remove_truck(hanomag)

# Autobusu izņemšanas pārbaude no nomas punkta.
rental_point.remove_bus(mercedes)
print()

# Testēšana, paziņojot par pieejamajām automašīnām.
```

```
rental_point.announce_cars()

# Testēšana, paziņojot par pieejamajām kravas auto.
rental_point.announce_trucks()

# Testēšana, paziņojot par pieejamiem autobusiem.
rental_point.announce_buses()

# Nomas automašīnu testēšana.
rental_point.sell_car(moskvich, buyer1)
rental_point.sell_car(pioneer, buyer2)
rental_point.sell_car(volga, buyer3)
rental_point.sell_car(zhiguli, buyer4)
rental_point.sell_car(moskvich, buyer5)
rental_point.sell_car(pioneer, buyer6)

# Kravas automašīnu nomas pārbaude.
rental_point.sell_car(hanomag, buyer1)
rental_point.sell_car(hummer, buyer2)
rental_point.sell_car(hanomag, buyer3)

# Pārbauda nomas autobusu pircēju.
rental_point.sell_car(mercedes, buyer4)
rental_point.sell_car(raf, buyer5)
rental_point.sell_car(mercedes, buyer6)

print()
buyer1.print_license()
buyer2.print_license()
buyer3.print_license()
buyer4.print_license()
```

```
buyer5.print_license()
buyer6.print_license()
buyer7.print_license()
print()

print("<Izvadīt tikai pieejamus kravas automašīnas>")
rental_point.announce_trucks()

print("<Izvadīt tikai pieejamus autobusus>")
rental_point.announce_buses()

print(str(volga.return_name()) + " ID:")
print(volga.id)
print(str(zhiguli.return_name()) + " ID:")
print(zhiguli.id)
print(str(moskvich.return_name()) + " ID:")
print(moskvich.id)
print(str(pioneer.return_name()) + " ID:")
print(pioneer.id)
print(str(hummer.return_name()) + " ID:")
print(hummer.id)
print(str(hanomag.return_name()) + " ID:")
print(hanomag.id)
print(str(mercedes.return_name()) + " ID:")
print(mercedes.id)
print(str(raf.return_name()) + " ID:")
print(raf.id)
print()

print("<Izvadīt tikai pieejamus vieglauto>")
rental_point.announce_cars()
```

rental_point.add_bus(mercedes)

rental_point.add_bus(raf)

rental_point.sell_car(mercedes, buyer1)

rental_point.sell_car(mercedes, buyer2)

rental_point.sell_car(mercedes, buyer4)

rental_point.sell_car(mercedes, buyer4)

rental_point.sell_car(moskvich, buyer4)

rental_point.return_vehicle(mercedes)

rental_point.return_vehicle(raf)

Testa piemēri:

1)

Pievienota fiziskā persona Artūrs Kariņš ar A kategorijas (motocikli) vadītāja apliecību.

Pievienota juridiskā persona SIA 'RVR' ar B kategorijas (vieglauto) vadītāja apliecību.

Pievienota fiziskā persona Edgars Pliekšāns bez vadītāja apliecību.

Pievienota juridiskā persona AS 'Gosbank' ar D kategorijas (autobuss) vadītāja apliecību.

Pievienota juridiskā persona SIA 'OGAS' bez vadītāja apliecību.

Pievienota fiziskā persona Lizete Rozenberga ar C kategorijas (kravas automašīnas) vadītāja apliecību.

Pievienota juridiskā persona SIA 'OBKhSS' ar B kategorijas (vieglauto) vadītāja apliecību.

Automašīna GAZ-24 Volga tika nogādāta nomas punktā!

Automašīna VAZ-2106 Zhiguli tika nogādāta nomas punktā!

Automašīna Moskvich-403 tika nogādāta nomas punktā!

Automašīna Pioneer 2M tika nogādāta nomas punktā!

Automašīna Lamborghini Diablo tika nogādāta nomas punktā!

Automašīna Rolls-Royce Phantom tika nogādāta nomas punktā!

Kravas automašīna Hummer H3 tika nogādata nomas punktā!

Kravas automašīna Hanomag-Henschel F55 tika nogādata nomas punktā!

Kravas automašīna UAZ-452 tika nogādata nomas punktā!

Autobuss Mercedes-Benz Sprinter tika nogādāts nomas punktā!

Autobuss RAF-2203 tika nogādāts nomas punktā!

Pašlaik nomai pieejami vieglauto:

GAZ-24 Volga

VAZ-2106 Zhiguli

Moskvich-403

Pioneer 2M

Lamborghini Diablo

Rolls-Royce Phantom

Pašlaik nomai pieejami autobusi:

Mercedes-Benz Sprinter

RAF-2203

Pašlaik nomai pieejami kravas auto:

Hummer H3

Hanomag-Henschel F55

UAZ-452

Nomas punkts nodeva Lamborghini Diablo metāllūžņos!

Nomas punkts nodeva Rolls-Royce Phantom metāllūžņos!

Nomas punkts nodeva kravas automašīnu Hummer H3 metāllūžņos!

Nomas punkts nodeva kravas automašīnu Hanomag-Henschel F55 metāllūžņos!

Nomas punkts nodeva autobusu Mercedes-Benz Sprinter metāllūžņos!

Pašlaik nomai pieejami vieglauto:

GAZ-24 Volga

VAZ-2106 Zhiguli

Moskvich-403

Pioneer 2M

Pašlaik nomai pieejami kravas auto:

UAZ-452

Pašlaik nomai pieejami autobusi:

RAF-2203

Artūrs Kariņš ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu vieglauto Moskvich-403.

Pioneer 2M tika nodots nomai! Tagad to noma SIA 'RVR'.

Edgars Pliekšāns ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu vieglauto GAZ-24 Volga.

AS 'Gosbank' ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu vieglauto VAZ-2106 Zhiguli.

Jāzeps Baumanis ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu vieglauto Moskvich-403.

Atvainojiet, SIA 'OGAS'! Pioneer 2M pašlaik nav pieejama automašīnas nomas punktā!

Atvainojiet, Artūrs Kariņš! Hanomag-Henschel F55 pašlaik nav pieejama automašīnas nomas punktā!

Atvainojiet, SIA 'RVR'! Hummer H3 pašlaik nav pieejama automašīnas nomas punktā!

Atvainojiet, Edgars Pliekšāns! Hanomag-Henschel F55 pašlaik nav pieejama automašīnas nomas punktā!

Atvainojiet, AS 'Gosbank'! Mercedes-Benz Sprinter pašlaik nav pieejams autobusu nomas punktā!

Jāzeps Baumanis ir nepieciešamas D kategorijas vadītāja apliecība, lai nomātu autobusu RAF-2203.

Atvainojiet, SIA 'OGAS'! Mercedes-Benz Sprinter pašlaik nav pieejams autobusu nomas punktā!

Artūrs Kariņš ir vadītāja apliecība kategorijā A.

SIA 'RVR' ir vadītāja apliecība kategorijā B.

Edgars Pliekšāns nav vadītāja apliecības.

AS 'Gosbank' ir vadītāja apliecība kategorijā D.

Jāzeps Baumanis nav vadītāja apliecības.

SIA 'OGAS' nav vadītāja apliecības.

Lizete Rozenberga ir vadītāja apliecība kategorijā C.

<Izvadīt tikai pieejamus kravas automašīnas>

Pašlaik nomai pieejami kravas auto:

UAZ-452

<Izvadīt tikai pieejamus autobusus>

Pašlaik nomai pieejami autobusi:

RAF-2203

GAZ-24 Volga ID:

0

VAZ-2106 Zhiguli ID:

1

Moskvich-403 ID:

2

Pioneer 2M ID:

3

Hummer H3 ID:

6

Hanomag-Henschel F55 ID:

7

Mercedes-Benz Sprinter ID:

9

RAF-2203 ID:

10

<Izvadīt tikai pieejamus vieglauto>

Pašlaik nomai pieejami vieglauto:

GAZ-24 Volga

VAZ-2106 Zhiguli

Moskvich-403

Autobuss Mercedes-Benz Sprinter tika nogādāts nomas punktā!

Autobuss RAF-2203 tika nogādāts nomas punktā!

Artūrs Kariņš ir nepieciešamas D kategorijas vadītāja apliecība, lai nomātu autobusu Mercedes-Benz Sprinter.

SIA 'RVR' ir nepieciešamas D kategorijas vadītāja apliecība, lai nomātu autobusu Mercedes-Benz Sprinter.

Mercedes-Benz Sprinter tika nodots nomai! Tagad to noma AS 'Gosbank'.

Atvainojiet, AS 'Gosbank'! Mercedes-Benz Sprinter pašlaik nav pieejams autobusu nomas punktā!

AS 'Gosbank' ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu vieglauto Moskvich-403.

Mercedes-Benz Sprinter tika atgriezts nomas punktā.

RAF-2203 nebija nomāts šajā nomas punktā!

2)

```
Pievienota fiziskā persona Artūrs Kariņš ar A kategorijas (motocikli) vadītāja apliecību.
Pievienota juridiskā persona SIA 'RVR' ar B kategorijas (vieglauto) vadītāja apliecību.
Pievienota fiziskā persona Edgars Pliekšāns bez vadītāja apliecību.
Pievienota juridiskā persona AS 'Gosbank' ar D kategorijas (autobuss) vadītāja apliecību.
Pievienota juridiskā persona SIA 'OGAS' bez vadītāja apliecību.
Pievienota fiziskā persona Lizete Rozenberga ar C kategorijas (kravas automašīnas) vadītāja apliecību.
Pievienota juridiskā persona SIA 'OBKhSS' ar B kategorijas (vieglauto) vadītāja apliecību.

Automašīna GAZ-24 Volga tika nogādāta nomas punktā!
Automašīna VAZ-2106 Zhiguli tika nogādāta nomas punktā!
Automašīna Moskvich-403 tika nogādāta nomas punktā!
Automašīna Pioneer 2M tika nogādāta nomas punktā!
Automašīna Lamborghini Diablo tika nogādāta nomas punktā!
Automašīna Rolls-Royce Phantom tika nogādāta nomas punktā!

Kravas automašīna Hummer H3 tika nogādata nomas punktā!
Kravas automašīna Hanomag-Henschel F55 tika nogādata nomas punktā!
Kravas automašīna UAZ-452 tika nogādata nomas punktā!

Autobuss Mercedes-Benz Sprinter tika nogādāts nomas punktā!
Autobuss RAF-2203 tika nogādāts nomas punktā!

Pašlaik nomai pieejami vieglauto:
GAZ-24 Volga
VAZ-2106 Zhiguli
Moskvich-403
Pioneer 2M
Lamborghini Diablo
Rolls-Royce Phantom

Pašlaik nomai pieejami autobusi:
Mercedes-Benz Sprinter
RAF-2203

Pašlaik nomai pieejami kravas auto:
Hummer H3
Hanomag-Henschel F55
UAZ-452
```

3)

```
<Izvadīt tikai pieejamus kravas automašīnas>
Pašlaik nomai pieejami kravas auto:
UAZ-452

<Izvadīt tikai pieejamus autobusus>
Pašlaik nomai pieejami autobusi:
RAF-2203

GAZ-24 Volga ID:
0
VAZ-2106 Zhiguli ID:
1
Moskvich-403 ID:
2
Pioneer 2M ID:
3
Hummer H3 ID:
6
Hanomag-Henschel F55 ID:
7
Mercedes-Benz Sprinter ID:
9
RAF-2203 ID:
10

<Izvadīt tikai pieejamus vieglauto>
Pašlaik nomai pieejami vieglauto:
GAZ-24 Volga
VAZ-2106 Zhiguli
Moskvich-403

Autobuss Mercedes-Benz Sprinter tika nogādāts nomas punktā!
Autobuss RAF-2203 tika nogādāts nomas punktā!
Artūrs Kariņš ir nepieciešamas D kategorijas vadītāja apliecība, lai nomātu autobusu Mercedes-Benz Sprinter.
SIA 'RVR' ir nepieciešamas D kategorijas vadītāja apliecība, lai nomātu autobusu Mercedes-Benz Sprinter.
Mercedes-Benz Sprinter tika nodots nomai! Tagad to noma AS 'Gosbank'.
Atvainojiet, AS 'Gosbank'! Mercedes-Benz Sprinter pašlaik nav pieejams autobusu nomas punktā!
AS 'Gosbank' ir nepieciešamas B kategorijas vadītāja apliecība, lai nomātu vieglauto Moskvich-403.
Mercedes-Benz Sprinter tika atgriezts nomas punktā.
RAF-2203 nebija nomāts šajā nomas punktā!
```

4)

```
Nomas punkts nodeva Lamborghini Diablo metāllūžņos!
Nomas punkts nodeva Rolls-Royce Phantom metāllūžņos!
Nomas punkts nodeva kravas automašīnu Hummer H3 metāllūžņos!
Nomas punkts nodeva kravas automašīnu Hanomag-Henschel F55 metāllūžņos!
Nomas punkts nodeva autobusu Mercedes-Benz Sprinter metāllūžņos!
```

5)

```
Artūrs Kariņš ir vadītāja apliecība kategorijā A.
SIA 'RVR' ir vadītāja apliecība kategorijā B.
Edgars Pliekšāns nav vadītāja apliecības.
AS 'Gosbank' ir vadītāja apliecība kategorijā D.
Jāzeps Baumanis nav vadītāja apliecības.
SIA 'OGAS' nav vadītāja apliecības.
Lizete Rozenberga ir vadītāja apliecība kategorijā C.
```

3*. uzdevums

Uzdevums par polimorfismu. Jums jāizveido programma, kas ļauj izveidot dažādas figūras un nosaka tās apkārtmēru un laukumu.

Kods:

```
# Programmas nosaukums: Dažādu figuru apkārtmērs un laukums ar polimorfismu.

# 3. uzdevums (1MPR12_Vladislavs_Babaņins)

# Uzdevuma formulējums: Uzdevums par polimorfismu. Jums jāizveido programma, kas ļauj izveidot dažādas figūras un nosaka tās apkārtmēru un laukumu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import math



class Taisnsturis:
    # Taisnstūris.
    def __init__(self, garums, platums):
        self.garums = garums
        self.platums = platums

    def perimetrs(self):
        # Aprēķina taisnstūra perimetru.
        return (self.garums + self.platums) * 2

    def laukums(self):
        # Aprēķina taisnstūra laukumu.
        return self.garums * self.platums



class Regular_pentagon:
```

```
    # Regulārs piecstūris.
    def __init__(self, side_length):
        self.side_length = side_length

    def perimetrs(self):
        # Aprēķina regulāra piecstūra perimetru.
        return 5 * self.side_length

    def laukums(self):
        # Aprēķina regulāra piecstūra laukumu.
        return ((math.sqrt(5) * math.sqrt(5 + 2 * math.sqrt(5))) / 4) * (self.side_length *
self.side_length)
        # Formulas avots:
        # https://en.wikipedia.org/wiki/Pentagon


class Regular_hexagon:
    # Regulārs sešstūris.
    def __init__(self, side_length):
        self.side_length = side_length

    def perimetrs(self):
        # Aprēķina regulāra sešstūra perimetru.
        return 6 * self.side_length

    def laukums(self):
        # Aprēķina regulāra sešstūra laukumu.
        return (3 * math.sqrt(3) / 2) * (self.side_length * self.side_length)


class Regular_polygon:
```

```
    # Regulārs n-stūris.
    def __init__(self, side_length, num_sides):
        # num_sides - n-stūra malu skaits (vai leņķu skaits) n-stūra n skaitlis.
        # side_lenght - vienas malas garums.
        self.side_length = side_length
        self.num_sides = num_sides

    def perimetrs(self):
        # Aprēķina regulāra n-stūra perimetru.
        return self.num_sides * self.side_length

    def laukums(self):
        # Aprēķina regulāra n-stūra laukumu.
        return 0.25 * self.num_sides * self.side_length * self.side_length / math.tan(math.pi /
self.num_sides)
        # Formulas avots:
        # https://en.wikipedia.org/wiki/Regular_polygon


class Trapece:
    def __init__(self, a, b, c, d):
        # Iniciālizē a, b, c, d.
        # a - trapeces pirmās malas garums.
        # b - trapeces otrais malas garums.
        # c - trapeces trešais malas garums.
        # d - trapeces trešais malas garums.
        # self - trapece.
        self.a = a
        self.b = b
        self.c = c
        self.d = d
```

```
    def perimetrs(self):
        # Aprēķina trijstūra perimetru.
        return self.a + self.b + self.c + self.d

    def laukums(self):
        # Aprēķina trapeces laukumu
        d = self.b - self.a
        k = (d * d + self.c * self.c - self.d * self.d) / (2 * self.c * d)
        k = k * k
        t = 1 - k
        m = math.sqrt(t)
        p = ((self.a + self.b) / 2) * self.c
        return p * m
        # Laukuma formulas avots:
        # https://math.stackexchange.com/questions/2637690/is-there-a-formula-to-calculate-the-area-of-a-trapezoid-knowing-the-length-of-al


class Trijsturis:
    # Trijstūra klase.
    def __init__(self, a, b, c):
        # Iniciālizē a, b, c.
        # a - trijstūra pirmās malas garums.
        # b - trijstūra otrais malas garums.
        # c - trijstūra trešais malas garums.
        # self - trijstūris.
        self.a = a
        self.b = b
        self.c = c
```

```
    def pusperimetrs(self):
        # Aprēķina trijstūra pusperimetru.
        p = (self.a + self.b + self.c) / 2
        return p

    def laukums(self):
        # Aprēķina trijstūra laukumu ar Herona formulu.
        p = Trijsturis.pusperimetrs(self)
        return math.sqrt(p * (p - self.a) * (p - self.b) * (p - self.c))  # math.sqrt(p * (p - self.a) * (p - self.b) * (p - self.c))  # self.garums * self.platums

    def perimetrs(self):
        # Aprēķina trijstūra perimetru.
        return self.a + self.b + self.c

    def ievilkta_rinka_radiuss(self):
        # Aprēķina trijstūra ievilkta riņķa līnijas rādiusu r.
        return Trijsturis.laukums(self) / Trijsturis.pusperimetrs(self)  # regulāram r = (a*sqrt(3))/6

    def apvilkta_rinka_radiuss(self):
        # Aprēķina trijstūra apvilkta riņķa līnijas rādiusu R.
        return self.a * self.b * self.c / (4 * Trijsturis.laukums(self))  # regulāram R = (a*sqrt(3))/6


class Rinkis:
    def __init__(self, radiuss):
        self.radiuss = radiuss

    def perimetrs(self):
        # Aprēķina riņķa līnijas perimetru (riņķa līnijas garumu) ar formulu pi*r*r = pi*r^2.
        return 6.283 * self.radiuss  # 2*pi*r
```

```
    def laukums(self):
        # Aprēķina riņķa līnijas laukumu ar formulu pi*r*r = pi*r^2.
        return 3.1415 * self.radiuss * self.radiuss  # pi*r*r = pi*r^2


# ----------------------------------------------------------
# Galvenā programmas daļa
# ----------------------------------------------------------


def izdrukat_laukumu(figura):
    print("Šīs figūras laukums ir:", figura.laukums())


def izdrukat_perimetru(figura):
    print("Šīs figūras perimetrs ir:", figura.perimetrs())


f1 = Rinkis(1)
f2 = Taisnsturis(1, 2)
f3 = Trijsturis(1, 1, 1)
f4 = Trapece(11, 6.403, 5.385, 5)
f5 = Regular_pentagon(1)
f6 = Regular_hexagon(1)
f7 = Regular_polygon(1, 8)
f8 = Regular_polygon(1, 3)
f9 = Regular_polygon(1, 10)
f10 = Regular_polygon(1, 4)

print("Riņķis ar rādiusu 1:")
```

```
izdrukat_perimetru(f1)
izdrukat_laukumu(f1)
print()

print("Taisnstūris(1,2):")
izdrukat_perimetru(f2)
izdrukat_laukumu(f2)
print()

print("Trijsturis(1, 1, 1):")
izdrukat_perimetru(f3)
izdrukat_laukumu(f3)
print()

print("Trapece(11, 6,403, 5,385, 5):")
izdrukat_perimetru(f4)
izdrukat_laukumu(f4)
print()

print("Regulārs piecstūris(1):")
izdrukat_perimetru(f5)
izdrukat_laukumu(f5)
print()

print("Regulārs sešstūris(1):")
izdrukat_perimetru(f6)
izdrukat_laukumu(f6)
print()

print("Regulārs 8-stūris(1):")
izdrukat_perimetru(f7)
```

```
izdrukat_laukumu(f7)
print()

print("Regulārs 3-stūris(1):")
izdrukat_perimetru(f8)
izdrukat_laukumu(f8)
print()

print("Regulārs 10-stūris(1):")
izdrukat_perimetru(f9)
izdrukat_laukumu(f9)
print()

print("Regulārs 4-stūris(1):")
izdrukat_perimetru(f10)
izdrukat_laukumu(f10)
print()
```

Testa piemēri:

1)

Riņķis ar rādiusu 1:

Šīs figūras perimetrs ir: 6.283

Šīs figūras laukums ir: 3.1415

Taisnstūris(1,2):

Šīs figūras perimetrs ir: 6

Šīs figūras laukums ir: 2

Trijsturis(1, 1, 1):

Šīs figūras perimetrs ir: 3

Šīs figūras laukums ir: 0.4330127018922193

Trapece(11, 6,403, 5,385, 5):

Šīs figūras perimetrs ir: 27.787999999999997

Šīs figūras laukums ir: 40.37242399037393

Regulārs piecstūris(1):

Šīs figūras perimetrs ir: 5

Šīs figūras laukums ir: 1.7204774005889671

Regulārs sešstūris(1):

Šīs figūras perimetrs ir: 6

Šīs figūras laukums ir: 2.598076211353316

Regulārs 8-stūris(1):

Šīs figūras perimetrs ir: 8

Šīs figūras laukums ir: 4.82842712474619

Regulārs 3-stūris(1):

Šīs figūras perimetrs ir: 3

Šīs figūras laukums ir: 0.43301270189221946

Regulārs 10-stūris(1):

Šīs figūras perimetrs ir: 10

Šīs figūras laukums ir: 7.694208842938134

Regulārs 4-stūris(1):

Šīs figūras perimetrs ir: 4

Šīs figūras laukums ir: 1.0000000000000002

2)

```
Rinķis ar rādiusu 1:
Šīs figūras perimetrs ir: 6.283
Šīs figūras laukums ir: 3.1415

Taisnstūris(1,2):
Šīs figūras perimetrs ir: 6
Šīs figūras laukums ir: 2

Trijsturis(1, 1, 1):
Šīs figūras perimetrs ir: 3
Šīs figūras laukums ir: 0.4330127018922193

Trapece(11, 6,403, 5,385, 5):
Šīs figūras perimetrs ir: 27.787999999999997
Šīs figūras laukums ir: 40.37242399037393

Regulārs piecstūris(1):
Šīs figūras perimetrs ir: 5
Šīs figūras laukums ir: 1.7204774005889671

Regulārs sešstūris(1):
Šīs figūras perimetrs ir: 6
Šīs figūras laukums ir: 2.598076211353316
```

3)

```
Regulārs 8-stūris(1):
Šīs figūras perimetrs ir: 8
Šīs figūras laukums ir: 4.82842712474619

Regulārs 3-stūris(1):
Šīs figūras perimetrs ir: 3
Šīs figūras laukums ir: 0.43301270189221946

Regulārs 10-stūris(1):
Šīs figūras perimetrs ir: 10
Šīs figūras laukums ir: 7.694208842938134

Regulārs 4-stūris(1):
Šīs figūras perimetrs ir: 4
Šīs figūras laukums ir: 1.0000000000000002
```