Kurs: Programiranje igara sa programskim jezikom Pajton
016 čas: Program Blekdžek.py
Teme: objekti iz različitih klasa, izvođenje novih klasa iz starih

Slanje i primanje poruka

Na neki način, OOP je poput ekosistema a objekti su poput organizama. Da bi se održao uspešan ekosistem, organizmi moraju imati interakciju. Isto važi i u OOP. Da bi program bio uspešan, objekat mora interagovati u detaljno definisanim načinima. Kaže se da objekti interaguju slanjem poruka jedni drugima. To znači da podižu jedni drugima metode.

program Vanzemaljski_blaster.py

Ovaj program simulira akcionu igru gde igrač uništava vanzemaljce koji daju pozdravni govor. To je ostvaruje kada jedan objekat šalje drugom objektu poruku. Tehnički, program instancira objekat Igrac, heroj, a objekat Vanzemaljac, osvajac. Kada herojev blast() metod se podigne sa osvajac kao njegovim argumentom, heroj podiže osvajac metodu smrt(). Može se reći da kada igrač uništi vanzemaljca, igrač šalje poruku vanzemaljcu kako ga je uništio.

U programiranju se koriste različiti dijagrami i metode za planiranje i mapiranje softverskih projekata. Jedan od najpopularnijih je Jedinstveni Jezik za Modelovanje (Unified Modeling Language (UML)) koji je notacioni jezik za vizuelizaciju OOP.

```
#Vanzemaljski_blaster
#Prikazuje interakciju objekata
class Igrac(object):
    """Igrac u akcionoj-pucackoj igri. """
    def blast(self, protivnik):
        print("Igrac je unistio protivnika.\n")
        protivnik.smrt()

class Vanzemaljac(object):
    """Vanzemaljac u akcionoj-pucackoj igri. """
    def smrt(self):
        print("Vanzemaljac uzdahnu i rece, 'Dakle, to je to."\
           "Ovo je ono pravo. \n" \
           "Da, postaje mracno. reci mojim 1.6 miliona larvi da sam ih voleo...\n"\
           "Zdravo, okrutni svete.")

print("\t\tSmrt Vanzemaljca\n")
heroj = Igrac()
osvajac = Vanzemaljac()
heroj.blast(osvajac)
```

```
                Smrt Vanzemaljca

Igrac je unistio protivnika.

Vanzemaljac uzdahnu i rece, 'Dakle, to je to.Ovo je ono pravo.
Da, postaje mracno. reci mojim 1.6 miliona larvi da sam ih voleo...
Zdravo, okrutni svete.
```

001 Slanje i primanje poruka

Pre nego jedan objekat pošalje poruku drugom objektu, treba napraviti dva objekata. U programu su to objekat heroj klase Igrac i objekat osvajac klase Vanzemaljac.

Sledeća linija koda kaže da kroz heroj.blast(osvajac), podiže se heroj metoda blast() i pridodaje osvajac (objekat Vanzemaljac) kao argument. Posmatrajući definiciju blast(), vidi se da metod prihvata objekat u svoj parametar protivnik. Tako da kada se blast() izvrši, protivnik ukazuje

na objekat Vanzemaljac. Posle prikaza poruke, blast() podiže od objekta Vanzemaljac, metod smrt() kroz protivnik.smrt(). U suštini, objekat Igrac šalje objektu Vanzemaljac poruku podizanjem njegove smrt() metode.
Objekat Vanzemaljac prima poruku od objakta Igrac u formi podizanja njegove metode smrt(). Objektova Vanzemaljac metoda smrt() zatim prikazuje string.

Kombinovanje objekata

U realnosti, objekti su sastavjeni od manjih nezavisnih objekata ili od kolekcije različitih objekata. Npr, može se napisati Zoo klasa koja ima atribute zivotinje, koja je lista različitih Zivotinja objekata. Kombinovanje objekata omogućava kreiranje kompleksnijih objekata od jednostavnijih.

program Igra_karatama.py

```
#Igra_kartama
#Prikazuje kombinovanje objekata
class Karta(object):
    """Igranje sa kartama. """
    VRSTA = ["A", "2", "3", "4", "5", "6", "7",
             "8", "9", "10", "J", "Q", "K"]
    ZNAK = ["t", "k", "h", "p"]

    def __init__(self, vrsta, znak):
        self.vrsta = vrsta
        self.znak = znak

    def __str__(self):
        tip = self.vrsta + self.znak
        return tip

class Ruka(object):
    """Karte koje su trenutno u ruci igraca. """
    def __init__(self):
        self.karte = []

    def __str__(self):
        if self.karte:
           tip = ""
           for karta in self.karte:
               tip += str(karta) + "  "
        else:
            tip = "<prazno>"
        return tip

    def prazno(self):
        self.karte = []

    def dodati(self, karta):
        self.karte.append(karta)

    def dati(self, karta, druga_ruka):
        self.karte.remove(karta)
        druga_ruka.dodati(karta)

karta1 = Karta(vrsta = "A", znak = "c")
print("Stampanje objekta Karta:")
print(karta1)

karta2 = Karta(vrsta = "2", znak = "c")
karta3 = Karta(vrsta = "3", znak = "c")
karta4 = Karta(vrsta = "4", znak = "c")
```

```
karta5 = Karta(vrsta = "5", znak = "c")
print("\nStampanje ostalih objekata pojedinacno:")
print(karta2)
print(karta3)
print(karta4)
print(karta5)

moja_ruka = Ruka()
print("\nStampanje moje ruke karata pre dodavanja karata:")
print(moja_ruka)

moja_ruka.dodati(karta1)
moja_ruka.dodati(karta2)
moja_ruka.dodati(karta3)
moja_ruka.dodati(karta4)
moja_ruka.dodati(karta5)
print("\nStampanje moje ruke posle dodavanja pet karata:")
print(moja_ruka)

tvoja_ruka = Ruka()
moja_ruka.dati(karta1, tvoja_ruka)
moja_ruka.dati(karta2, tvoja_ruka)
print("\nDati prve dve karte iz moje ruke u tvoju ruku.")
print("Tvoja ruka:")
print(tvoja_ruka)
print("Moja ruka:")
print(moja_ruka)

moja_ruka.prazno()
print("\nMoja ruka posle davanja karata:")
print(moja_ruka)
```

```
Stampanje objekta Karta:
Ac

Stampanje ostalih objekata pojedinacno:
2c
3c
4c
5c

Stampanje moje ruke karata pre dodavanja karata:
<prazno>

Stampanje moje ruke posle dodavanja pet karata:
Ac  2c  3c  4c  5c

Dati prve dve karte iz moje ruke u tvoju ruku.
Tvoja ruka:
Ac  2c
Moja ruka:
3c  4c  5c

Moja ruka posle davanja karata:
<prazno>
```

002 Kreiranje klase Karta

Prvo se kreira klasa Karta za objekte koji predstavljaju karte za igranje. Svaki objekat Karta ima vrsta atribut, koji predstavlja simbol i vrednost karte. Moguće vrednosti su izlistane u atributu klase VRSTA.

```
class Karta(object):
    """Igranje sa kartama. """
    VRSTA = ["A", "2", "3", "4", "5", "6", "7",
             "8", "9", "10", "J", "Q", "K"]
    ZNAK = ["t", "k", "h", "p"]

    def __init__(self, vrsta, znak):
        self.vrsta = vrsta
        self.znak = znak

    def __str__(self):
        tip = self.vrsta + self.znak
        return tip
```

Svaka karta takođe ima znak atribut, koji predstavlja oznaku karte. Moguće vrednosti za ovaj atribut su izlistane u atribut klasi ZNAK, gde je t – tref, h – herc, p – pik, k - karo. Sada, objekat sa atributom vrsta “A“ i atributom znak “k“ je as karo.

Poseban metod __str_() samo vraća nadovezivanje vrste i znak atributa tako da se objekat može odštampati.

003 Kreiranje klase Ruka

Sledeće u programu se kreira klasa Ruka za objekte, koji su kolekcija Karta objekata:

```
class Ruka(object):
    """Karte koje su trenutno u ruci igraca. """
    def __init__(self):
        self.karte = []

    def __str__(self):
        if self.karte:
           tip = ""
           for karta in self.karte:
               tip += str(karta) + "  "
        else:
            tip = "<prazno>"
        return tip

    def prazno(self):
        self.karte = []

    def dodati(self, karta):
        self.karte.append(karta)

    def dati(self, karta, druga_ruka):
        self.karte.remove(karta)
        druga_ruka.dodati(karta)
```

Novi Ruka objekat ima atribut karte koji je namenjen da bude lista Karta objekata. Tako da svaki Ruka objekat ima atribut koji je lista mogućih mnogo drugih objekata.

Poseban metod __str__() vraća string koji predstavlja celu ruku. Metoda iterira kroz svaki Karta objekat u Ruka objektu i nadovezuje string reprezentaciju Karta objekata. Ako Ruka objekat nema Karta objekte, string “<prazno>“ se vraća.

Metoda prazno() čisti listu od karata dodeljivanjem prazne liste atributu karte objekta.

Metod dodati() dodaje objekat atributu karte.

Metod dati() odstranjuje objekat iz Ruka objekta i pridodaje ga na drugi Ruka objekat podizanjem dodati() metode drugog Ruka objekta. Drugi način da se to objasni je da prvi Ruka objekat šalje drugom Ruka objektu poruku za dodavanje objekta Karta.

004 Korišćenje Karta objekata

U glavnom delu programa, kreirana su i odštampana pet Karta objekata:

```
karta1 = Karta(vrsta = "A", znak = "t")
print("Stampanje objekta Karta:")
print(karta1)

karta2 = Karta(vrsta = "2", znak = "t")
karta3 = Karta(vrsta = "3", znak = "t")
karta4 = Karta(vrsta = "4", znak = "t")
karta5 = Karta(vrsta = "5", znak = "t")
print("\nStampanje ostalih objekata pojedinacno:")
print(karta2)
print(karta3)
print(karta4)
print(karta5)
```

Prvi Karta objekat koji je kreiran ima vrsta atribut jednak sa “A“ i znak atribut jednak sa “t“. Kada se objekat odštampa, on je prikazan na ekranu kao At. Ostali objekti su prikazani na sličan način.

005 Kombinovanje Karta objekata korišćenjem Ruka objekta

Kreira se Ruka objekat, dodeljuje se u moja_ruka promenjivu a zatim i štampa:

```
moja_ruka = Ruka()
print("\nStampanje moje ruke karata pre dodavanja karata:")
print(moja_ruka)
```

Pošto je objektov atribut karte prazna lista, štampanje objekta prikazuje se tekst <prazno>.

Dalje se dodaje pet Karta objekata u moja_ruka i opet štampa:

```
moja_ruka.dodati(karta1)
moja_ruka.dodati(karta2)
moja_ruka.dodati(karta3)
moja_ruka.dodati(karta4)
moja_ruka.dodati(karta5)
print("\nStampanje moje ruke posle dodavanja pet karata:")
print(moja_ruka)
```

Ovog puta, tekst At 2t 3t 4t 5t se pojavljuje.

Zatim se kreira novi Ruka objekat, tvoja_ruka. Korišćenjem metode dati() od moja_ruka, vrši se transfer prve dve karte iz moja_ruka u tvoja_ruka. Zatim se obe ruke štampaju:

```
tvoja_ruka = Ruka()
moja_ruka.dati(karta1, tvoja_ruka)
moja_ruka.dati(karta2, tvoja_ruka)
print("\nDati prve dve karte iz moje ruke u tvoju ruku.")
print("Tvoja ruka:")
print(tvoja_ruka)
print("Moja ruka:")
print(moja_ruka)
```

vidi se da tvoja_ruka sadrži At 2t, dok moja_ruka ima 3t 4t 5t.

Korišćenje Nasleđivanja za kreiranje novih klasa

Jedan od glavnih elemenata OOP je nasleđivanje (inheritance), koje oogućava zasnivanje nove klase na postojećim klasama. Na taj način, nova klasa automatski dobija (ili nasleđuje) sve metode i atribute postojeće klase.

U Pajtonu je moguće kreirati novu klasu koja direktno nasleđuje iz više od jedne klase. To se naziva višestruku nasleđivanje (multiple inheritance). Ali na taj način se pojavljuju i višestruke komplikacije što nije tema za početnike programere.

006 Proširenje klase kroz nasleđivanje

Nasleđivanje je posebno korisno kada se želi napraviti još više specijalizovana verzija postojeće klase. Preko nasleđivanja od postojeće klase, nova klasa dobija sve metode i atribute postojeće klase. Ali takođe se mogu dodati metode i atributi za novu klasu čime se proširuju mogućnosti objekata nove klase.

program Igra_kartama2.py

```
#Igra_kartama2
#Prikazuje nasledjivanje - prosirenje klase
class Karta(object):
    """Igranje sa kartama. """
    VRSTA = ["A", "2", "3", "4", "5", "6", "7",
             "8", "9", "10", "J", "Q", "K"]
    ZNAK = ["t", "k", "h", "p"]

    def __init__(self, vrsta, znak):
        self.vrsta = vrsta
        self.znak = znak

    def __str__(self):
        tip = self.vrsta + self.znak
        return tip

class Ruka(object):
    """Karte koje su trenutno u ruci igraca. """
    def __init__(self):
        self.karte = []

    def __str__(self):
        if self.karte:
           tip = ""
           for karta in self.karte:
               tip += str(karta) + "\t"
        else:
            tip = "<prazno>"
        return tip

    def prazno(self):
        self.karte = []

    def dodati(self, karta):
        self.karte.append(karta)

    def dati(self, karta, druga_ruka):
        self.karte.remove(karta)
        druga_ruka.dodati(karta)

class Spil(Ruka):
    """ Spil karata. """
    def popuniti(self):
        for znak in Karta.ZNAK:
            for vrsta in Karta.VRSTA:
                self.dodati(Karta(vrsta, znak))

    def mesati(self):
        import random
```

```
        random.shuffle(self.karte)

    def podeliti(self, ruke, po_ruci = 1):
        for potezi in range(po_ruci):
            for ruka in ruke:
                if self.karte:
                    karta_na_vrhu = self.karte[0]
                    self.dati(karta_na_vrhu, ruka)
                else:
                    print("Deljenje se ne moze nastaviti. Nema vise karata!")

spil1 = Spil()
print("Napravljen je novi spil.")
print("Spil:")
print(spil1)

spil1.popuniti()
print("\nPopunjavanje spila.")
print("Spil:")
print(spil1)

spil1.mesati()
print("\nMesanje spila.")
print("Spil:")
print(spil1)

moja_ruka = Ruka()
tvoja_ruka = Ruka()
ruke = [moja_ruka, tvoja_ruka]
spil1.podeliti(ruke, po_ruci = 5)
print("\nPodeljeno je po 5 karata mojoj i tvojoj ruci.")
print("Moja ruka:")
print(moja_ruka)
print("Tvoja ruka:")
print(tvoja_ruka)
print("Spil:")
print(spil1)

spil1.prazno()
print("\nOciscen je spil.")
print("Spil:", spil1)
```

```
Napravljen je novi spil.
Spil:
<prazno>

Popunjavanje spila.
Spil:
At  2t  3t  4t  5t  6t  7t  8t  9t  10t Jt  Qt  Kt  Ak  2k  3k  4k  5k
6k  7k  8k  9k  10k Jk  Qk  Kk  Ah  2h  3h  4h  5h  6h  7h  8h  9h  10h
Jh  Qh  Kh  Ap  2p  3p  4p  5p  6p  7p  8p  9p  10p Jp  Qp  Kp

Mesanje spila.
Spil:
Ah  Ap  10t 10p Qk  Jk  7k  8p  2k  3t  Kp  Qt  Jt  9t  Jp  8k  7h  2h
10k 9h  5k  Kh  10h Qp  9k  Ak  Kt  7t  Kk  3h  4p  9p  2p  6k  5p  8h
4t  2t  4k  8t  4h  6h  Qh  6p  5h  3p  5t  3k  6t  Jh  7p  At

Podeljeno je po 5 karata mojoj i tvojoj ruci.
Moja ruka:
Ah  10t Qk  7k  2k
Tvoja ruka:
Ap  10p Jk  8p  3t
Spil:
Kp  Qt  Jt  9t  Jp  8k  7h  2h  10k 9h  5k  Kh  10h Qp  9k  Ak  Kt  7t
Kk  3h  4p  9p  2p  6k  5p  8h  4t  2t  4k  8t  4h  6h  Qh  6p  5h  3p
5t  3k  6t  Jh  7p  At

Ociscen je spil.
Spil: <prazno>
```

Ovaj program je baziran na programu Igra_kartama.py. Nova verzija unkorporira klasu Spil za opisivanje špila karata za igranje. Ipak, za razliku od ostalih klasa, klasa Spil je bazirana na postojećeoj klasi, Ruka. Kao rezultat, Spil automatski nasleđuje sve metode klase Ruka. Klasa Spil je kreirana pošto je špil karata poput specijalizovane ruke sa kartama. To jeste ruka ali sa posebnim ponašanjem. Špil može raditi isto što i ruka. To je kolekcija karata. Može dati kartu drugoj ruci,itd. Još uz to, špil može uraditi nekoliko stvari koje ruka ne može. Špil može biti promešan i može podeliti karte na dve različite ruke.

007 Nasleđivanje iz osnovne klase

Na početku programa, klase Kart i Ruka su iste kao u prethodnom programu.
Sledeće se kreira klasa Spil. Vidi se po hederu klase da je Spil baziran na klsai ruka:
Class Spil(Ruka):
Ruka je osnovna klasa (base class) pošto je Spil baziran na njoj. Spil se smatra izvedenom klasom (derived class) pošto izvodi deo svojih definicija iz Ruka klase. Kao rezultat ovog odnosa, Spil nasleđuje sve Ruka metode. Čak i da nije definisao ni jednu novu metodu u svojoj klasi, Spil objekat bi opet imao sve matode definisane u Ruka klasi: __init__(), __str__(), prazno(), dodati(), dati().

008 Proširivanje izvedene klase

U glavnom delu programa prvo se instancira novi Spil objekat:
spil1 = Spil()
Ako se pogleda u klasu Spil, vidi se da nije definisan metod konstruktor u klasi. Ali Spil nasleđuje Ruka konstruktor, tako da metod je automatski podignut sa novo kreiranim Spil objektom. Kao rezultat, novi Spil objekat dobija karte atribut koji je inicijalizovan na praznu listu, baš kao što bi i novo kreirani Ruka objekat dobio sličan karte atribut. Na kraju, operator dodele dodeljuje nov objekat u spil1.
Tako kreirani novi špil se štampa:

```
print("Napravljen je novi spil.")
print("Spil:")
print(spil1)
```

Nije definisan poseban __str__() metod u Spil takođe, ali opet, Spil nasleđuje metod iz Ruka. Pošto je spil prazan, kod prikazuje tekst `<prazno>` . Do sada, špil izgleda isto kao i ruka. To je zato što špil je specijalizovana vrsta ruke.
Podignut je objektov metod popuniti(), koji popunjava špil sa klasične 52 karte:
spil1.popuniti()
Sada špil ima nešto što ruka nema. To je zato što popuniti() metoda je nova metoda koja se definiše u Spil klasi. Metoda popuniti() iterira kroz sve 52 moguće kombinacije vrednosti u Karta.VRSTA i Karta.ZNAK (po jedna za svaku kartu u špilu). Za svaku kombinaciju, metod kreira nov Karta objekat koji se dodaje u špil.
Onda se štampa špil:

```
print("\nPopunjavanje spila.")
print("Spil:")
print(spil1)
```

Ovoga puta, sve 52 karte se prikazuju i to u uređenom redosledu. Zatim se karte promešaju:

```
spil1.mesati()
```

Definiše se metoda mesati() u Spil. Ovde se importuje modul random a zatim se poziva random.shuffle() funkcija sa objektovim karte atributom. Metoda random.shuffle() promeša elemente u listi po slučajnom redosledu. Tako da svi elementi od karte su promešani.

```
print("\nMesanje spila.")
print("Spil:")
print(spil1)
```
Zatim se kreiraju dva Ruka objekta i stave se na listu koja se dodeljuje u ruke:
```
moja_ruka = Ruka()
tvoja_ruka = Ruka()
ruke = [moja_ruka, tvoja_ruka]
```
Zatim se sevakoj ruci podeli po pet karata:
```
spil1.podeliti(ruke, po_ruci = 5)
```
Metod podeliti() je nov metod koji je definisan u Spil. Ima dva argumenta: listu ruka i broj karata koji se dele svakoj od ruka. Metod daje kartu iz špila svakoj ruci. Ako je špil ostao bez karata, metod štampa poruku. Metod ponavlja proces sve dok ne podeli potreban broj karata svakoj od ruka. Dakle, prethodna linija koda deli pet karata iz spil1 svakoj od ruka.
Da bi se video rezultat deljeenja, štampa se sadržaj svake od ruka i špil:
```
print("\nPodeljeno je po 5 karata mojoj i tvojoj ruci.")
print("Moja ruka:")
print(moja_ruka)
print("Tvoja ruka:")
print(tvoja_ruka)
print("Spil:")
print(spil1)
```
Promena ponašanja nasleđenih metoda
Do sada je prikazano kako se proširuje klasa dodavanjem novih metoda u izvedenu klasu. Ali, takođe se može redefinisati kako nasleđena metoda iz osnovne klase funkcioniše u izvedenoj klasi. Ovo se naziva **prepravljanje** (overriding) metode. Kada se prepravi metoda osnovne klase, postoje dve mogućnosti. Može se napraviti metoda sa potpuno novom funkcionalnosti ili se može uključiti funkcionalnost metode osnovne klase koja se prepravlja.
program Igra_kartama3.py
U ovom programu se izvode dve nove klase karata za igranje iz Karta klase koja se do sada koristila. Prva nova klasa definiše karte koje se ne mogu štampati. Tačnije, kada se štampa objekat ove klase, tekst <ne moze se stampati> se pojavi. Sledeća klasa definiše karte koje mogu biti ili okrenute nagore (face up) ili okrenute nadole (face down). Kada se štampa objekat ove klase, postoje dva moguća rezultata. Ako je karta okrenuta nagore, štampa je kao i svaki objekat klase Karta. Ako je karta okrenuta nadole, tekst XX se prikazuje.
```
#Igra_kartama3
#Prikazuje nasledjivanje - ponistenje metoda
class Karta(object):
    """ Igranje sa kartama. """
    VRSTA = ["A", "2", "3", "4", "5", "6", "7",
             "8", "9", "10", "J", "Q", "K"]
    ZNAK = ["t", "k", "h", "p"]

    def __init__(self, vrsta, znak):
        self.vrsta = vrsta
        self.znak = znak

    def __str__(self):
        tip = self.vrsta + self.znak
        return tip

class Nestampana_Karta(Karta):
    """ Karta koja ne pokazuje svoj znak i vrstu kada se stampa. """
    def __str__(self):
        return "<ne moze se odstampati>"
```

```
class Pozicionisana_Karta(Karta):
    """ Karta koja moze biti okrenuta nagore ili nadole. """
    def __init__(self, vrsta, znak, okrenuta_nagore = True):
        super(Pozicionisana_Karta, self).__init__(vrsta, znak)
        self.jeste_okrenuta_nagore = okrenuta_nagore

    def __str__(self):
        if self.jeste_okrenuta_nagore:
            tip = super(Pozicionisana_Karta, self).__str__()
        else:
            tip = "XX"
        return tip

    def obrnuti(self):
        self.jeste_okrenuta_nagore = not self.jeste_okrenuta_nagore

karta1 = Karta("A", "t")
karta2 = Nestampana_Karta("A", "k")
karta3 = Pozicionisana_Karta("A", "h")

print("Stampanje Karta objekta:")
print(karta1)

print("\nStampanje objekta Nestampana_Karta:")
print(karta2)

print("\nStampanje objekta Pozicionisana_Karta:")
print(karta3)
print("Obrtanje objekta Pozicionisana_Karta.")
karta3.obrnuti()
print("Stampanje objekta Pozicionisana_Karta:")
print(karta3)
```

```
Stampanje Karta objekta:
At

Stampanje objekta Nestampane_Karta:
<ne moze se odstampati>

Stampanje objekta Pozicionisana_Karta:
Ah
Obrtanje objekta Pozicionisana_Karta.
Stampanje objekta Pozicionisana_Karta:
XX
```

009 Prepravljanje metoda osnovne klase

Program koristi poznatu Karta klasu.

Dalje, izvodi se nova klasa za karte koje se ne mogu štampati bazirane na Karta. Heder klase:

```
class Nestampana_Karta(Karta):
```

Od ovoga hedera zna se da Nestampana_Karta nasleđuje sve metode iz Karta. Ali ponašanje nasleđenog metoda se može promeniti njegovim definisanjem unutar izvedene klase. To se i uradilo sa ostatkom definicije metode:

```
""" Karta koja ne pokazuje svoj znak i vrstu kada se stampa. """
    def __str__(self):
        return "<ne moze se odstampati>"
```

Klasa Nestampana_Karta nasleđuje __str__() metod iz Karta. Ali , definisan je nov __str__() metod u Nestampana_Karta koji prepravlja (zamenjuje) nasleđenu metodu. Svaki put kada se kreira metoda u izvedenoj klasi sa istim imenom kao i nasleđeni metod, vrši se prepravljanje nasleđenog metoda u novoj klasi. Tako da, kada se odštampa Neštampana_Karta objekat, ispisuje se poruka.

Izvedena klasa nema efekat na osnovnu klasu. Osnovna klasa i dalje funkcioniše kako je prethodno definisana.

010 Podizanje metoda osnovne klase

Ponekad kada se izvrši prepravljanje metode osnovne klase, poželjno je unkljućiti nasleđenu funkcionalnost metode. Npr, treba napraviti novi tip klase karata za igranje bazirano na klasi Karta. Treba da objekat ove nove klase da ima atribut koji ukazuje da li je karat okrenuta nagore ili ne. To znači da treba prepraviti nasleđenu metodu konstruktor iz Karta sa novim konstruktorom koji kreira atribut za proveru da li je okrenut nagore. Ipak, takođe treba da taj novi konstruktor kreira i postavi vrsta i znak atribute, kao što to već radi Karta konstruktor. Umesto ponovnog kucanja koda iz Karta konstruktora, može se on podići iz unutrašnjosti novog konstruktora. Zatim, reši se problem kreiranja i inicijalizacije vrsta i znak atributa za objekat nove klase. U konstruktoru nove klase, može se dodati atribut koji ukazuje da li je karta okrenuta nagore. To je tačno i pristup koji se koristi u klasi Pozicionisana_Karta:

```
class Pozicionisana_Karta(Karta):
    """ Karta koja moze biti okrenuta nagore ili nadole. """
    def __init__(self, vrsta, znak, okrenuta_nagore = True):
        super(Pozicionisana_Karta, self).__init__(vrsta, znak)
        self.jeste_okrenuta_nagore = okrenuta_nagore
```

Nova funkcija u konstruktoru, super(), omogućava podizanje metoda osnovne klase (takođe se zove i **superklasa**). Linija `super(Pozicionisana_Karta, self).__init__(vrsta, znak)` Podiže __init__() metodu od Karta (superklasa od Pozicionisana_Karta). Prvi argument u pozivu prema super(), Pozicionisana_Karta, kaže da treba podići metodu superklase (ili osnovne klase) Poziciona_Karta, koja je Karta. Sledeći argument, self, upućuje referencu na novo instancirani Pozicionisana_Karta objekat tako da kod u Karta može dobiti objekat da bi dodao vrta i znak atribute u njega. Sledeći deo iskaza, __init__(vrsta, znak), kaže Pajtonu da treba podići metod konstruktor iz Karta i da mu treba priključiti vrednosti vrsta i znak.

Sledeći metod u Pozicionisana_Karta takođe prepravlja metodu nasleđenu iz Karta i podiže prepravljenu metodu:

```
def __str__(self):
        if self.jeste_okrenuta_nagore:
            tip = super(Pozicionisana_Karta, self).__str__()
        else:
            tip = "XX"
        return tip
```

Metoda __str__() prvo proverava da li je objektov atribut okrenut_nagore True (što znači da je karta okrenuta nagore). Ako je tako, string predstavljanje za karte je postavljena na string vraćen iz Karta __str__() metodu pozvanu sa Pozicionisana_Karta objektom. Drugim rečima, ako je karta okrenuta nagore, karta štampa bilo koji objekat iz klase Karta. Ipak, ako karta nije okrenuta nagore, string reprezentacija koja se vrati je “XX”.

Poslednji metod u klasi ne prepravlja nasleđenu metodu. On samo proširuje definiciju nove klase:

```
def obrnuti(self):
        self.jeste_okrenuta_nagore = not self.jeste_okrenuta_nagore
```

Metod obrće kartu preokretanjem vrednosti objektovog okrenuta_nagore atributa. Ako objektov okrenuta_nagore atribut je True, onda podizanje objektove obrnuti() metode postavlja atribut na False. Ako je objektov okrenuta_nagore atribut False, onda podizanjem objektove obrnuti() metode se postavlja atribut na True.

011 Upotreba izvedenih klasa

U glavnom delu programa kreirana su tri objekta: jedan iz Karte, drugi iz Nestampana_Karta i poslednji iz Pozicionisana_Karta:

```
karta1 = Karta("A", "t")
karta2 = Nestampana_Karta("A", "k")
karta3 = Pozicionisana_Karta("A", "h")
```

Zatim se štampa objekat Karta:

```
print("Stampanje Karta objekta:")
print(karta1)
```

Ovo funkcioniše kao i u prethodnim programima i tekst At je prikazan.

Zatim se štampa objekat Nestampana_Karta:

```
print("\nStampanje objekta Nestampana_Karta:")
print(karta2)
```

Iako objekat ima atribut vrsta postavljen na “A” i znak atribut postavljen na “k”, štampanje objekta prikazuje tekst <ne moze se odstampati> pošto klasa Nestampana_Karta prepravlja njenu nasleđenu __str__() metodu sa jednom koja uvek vraća string <ne moze se odstampati>.

Sledeće dve linije štampaju objekat Pozicionisana_Karta:

```
print("\nStampanje objekta Pozicionisana_Karta:")
print(karta3)
```

Pošto objektov okrenuta_nagore atribut je True, objektov __str__() metoda podiže Karta __str__() metodu i tekst Ah se prikazuje.

Sledeće, podiže se Pozicionisana_Karta objektov obrnuti() metoda:

```
print("Obrtanje objekta Pozicionisana_Karta.")
karta3.obrnuti()
```

Kao rezultat, objektov okrenuti_nagore atribut je postavljen na False.

Sledeće dve linije štampaju Pozicionisana_Karta objekat ponovo:

```
print("Stampanje objekta Pozicionisana_Karta:")
print(karta3)
```

Sada je XX prikazano pošto objektov okrenuti_nagore atribut je False.

Polimorfizam

Polimorfizam je sposobnost da se različiti tipovi stvari posmatraju na isti način i da te stvari reaguju na sopstveni način. U kontekstu OOP, polimorfizam znači da se može poslati ista poruka na objekte različitih klasa koji su povezani po nasleđivanju i dostigli različite i odgovarajuće rezultate. Npr, Nestampana_Karta je izvedena iz Karta, a kada se podigne __str_() metod za objekat Nestampana_Karta, dobijaju se različiti rezultati nego kada se podigne __str__() metod iz objekta Karta. Rezultat polimorfizma ponašanja je taj da se može odštampati objekat čak i kada se ne zna da li je iz Nestampana_Karta ili Karta objekta. Bez obzira na kalsu objekta, kada se štampa, njegova metoda __str__() se podiže i tačna reprezentacija stringa se prikazuje.

Kreiranje modula

Moćan deo programiranja u Pajtonu je da se mogu kreirati, koristiti i deliti sopstveni moduli.

Kreiranjem sopstvenih modula (korisničkih) moduli se mogu više puta ponovo upotrebiti, što štedi vreme i uložni trud. Npr, mogu se ponovo korisiti Karta, Ruka i Spil klase za kreiranje drugih tipova igrara sa kartama bez potreba za ponovnim definisanjem funkcionalnost koje pružaju ove tri klase.

Razlaganjem programa na logičke module, veliki programi postaju lakši za upravljanje. Dosadašnji programi su bili održavani u jednom fajlu. Pošto su dosta kratki, to nije ništa dramatično. Ali ako se program sastoji od nekoliko hiljada linija koda, jedan veliki fajl bi dao velike probleme u upravljenju. Podela koda na module takođe čini lakšim za članove softverskih timova da se usredsrede na odvojene probleme unutar jednog projekta.

Kreiranjem modula, može se deliti kod drugim korisnicima na upotrebu. Ako se napravi koristan modul, može se poslati na e-mail koji ga može koristiti kao bilo koji drugi ugrađeni modul.

program Prosta_igra.py

Program traži broj igrača i ime svakog od njih. Zatim, program dodeljuje slučajno generisan skor za svakoga i prikazuje rezultate. Program koristi nov modul sa funkcijama i klasama.

012 Pisanje modula

Modul se piše kao i svaki drugi program u Pajtonu. Kada se napravi modul treba napraviti kolekciju povezanih programskih komponenti, poput funkcija i klasa, i sve smestiti u jedan fajl da bi bili importovani u novi program.

Kreiran je osnovni modul, nazvan igre, koji sadrži sve funkcije i klasu koje mogu biti korisne pri kreiranju igre:

```
#Igre
#Prikazuje kreiranje modula
class Igrac(object):
    """ Igrac za igru. """
    def __init__(self, ime, rezultat = 0):
        self.ime = ime
        self.rezultat = rezultat

    def __str__(self):
        tip = self.ime + ":\t" + str(self.rezultat)
        return tip

    def pitanje_da_ne(pitanje):
        """Postavlja da/ne pitanje."""
        odgovor = None
        while odgovor not in ("d", "n"):
            odgovor = input(pitanje).lower()
            return odgovor

    def trazi_broj(pitanje, nisko, visoko):
        """Trazi broj unutar opsega."""
        odgovor = None
        while odgovor not in range(nisko, visoko):
            odgovor = int(input(pitanje))
            return odgovor
```

Modul je nazvan igre pošto je sačuvan u fajlu igre.py. Imena modula su zasnovana na imenu fajla u kojem su sačuvani.

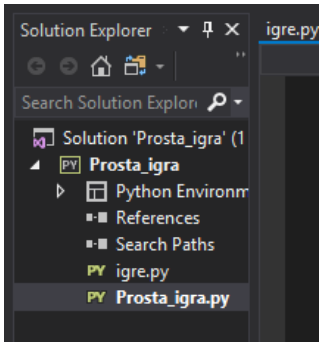
Klasa Igrac definiše objekat sa dva atributa, ime i rezultat, koji su postavljeni u metodi konstruktora. Postoji samo još jedan metod, __str__() koji vraća string reprezentaciju tako da objekat može biti odštampan.

Sledeća dve funkcije su se i ranije koristile, pitanje_da_ne() i trazi_broj().

013 Importovanje modula

```
#Prosta_igra
#Prikazuje importovanje modula
import igre, random
print("Dobrodosli u najprostiju igru na svetu!\n")
ponovo = None
while ponovo != "n":
    igraci = []
    broj = igre.Igrac.trazi_broj(pitanje = "Koliko igraca? (2 - 5): ",
                                 nisko = 2, visoko = 5)
    for i in range(broj):
        ime = input("Ime igraca: ")
        rezultat = random.randrange(100) + 1
        igrac = igre.Igrac(ime, rezultat)
        igraci.append(igrac)

    print("\nOvo su rezultati igre:")
    for igrac in igraci:
        print(igrac)

    ponovo = igre.Igrac.pitanje_da_ne("\nDa li zelis ponovo da igras? (d/n): ")
```

```
Dobrodosli u najprostiju igru na svetu!

Koliko igraca? (2 - 5): 3
Ime igraca: ja
Ime igraca: ti
Ime igraca: ona

Ovo su rezultati igre:
ja:     10
ti:     87
ona:    7

Da li zelis ponovo da igras? (d/n): n
```



U Visual Studio 2017 aplikaciji, prvo se kreira nov projekat koji nosi ime fajla u kojem se planira da bude glavni (main) deo programa. Zatim se desni klik na Solution Explorer, stavku Prosta_igra (startni fajl, ne Prosta_igra.py) dobija padajući meni u kome se izabere Add->New Item.

Tako se otvara novi .py fajl kome se prvo da potrebno ime (ovde je igre) i to će biti novi modul unutar projekta. Modul će se pojaviti na spisku korišćenih fajlova projekta unutar Solution Explorer-a.

Prvo se importuje programerski kreiran modul na isti način na koji se importuje ugrađeni modul. Problem može predstavlajti ako se importovani modul ne nalazi u istom direktorijumu gde je i fajl sa main programom koji ga i importuje.

014 Upotreba importovanih funkcija i klasa

Prvo se pita koliko igrača će učestvovati u igri:

```
ponovo = None
while ponovo != "n":
    igraci = []
    broj = igre.Igrac.trazi_broj(pitanje = "Koliko igraca? (2 - 5): ",
```

```
                                nisko = 2, visoko = 5)
```

Pozivanjem funkcije trazi_broj() iz igre modula dobija se broj igrača koji će učestvovati u igri. Za to se koristi dot notacija, gde se prvo specificira ime modula, ime klase unutar modula a zatim i naziv funkcije unutar klase.

Za svakog igrača, dobija se ime igrača i generiše slučajan broj bodova između 1 i 100 pozivanjem randrange() funkcije iz random modula. Zatim, kreira se objekat igrač korišćenjem ovog imena i rezultata. Pošto klasa Igrač je definisana u igre modulu, koristi se dot notacija i uključuje ime modula pre imena klase. Zatim se pridodaje ovaj novi objekat igrač u listu igrača.

```
for i in range(broj):
        ime = input("Ime igraca: ")
        rezultat = random.randrange(100) + 1
        igrac = igre.Igrac(ime, rezultat)
        igraci.append(igrac)
```

Zatim se štampa svaki igrač u igri:

```
print("\nOvo su rezultati igre:")
    for igrac in igraci:
        print(igrac)
```

Funkcija pitanje_da_ne() dobija odgovor na pitanje da li će igra da se nastavi:

```
ponovo = igre.Igrac.pitanje_da_ne("\nDa li zelis ponovo da igras? (d/n): ")
```

program Blekdzek.py

Za ovu igru je kreiran karte modul baziran na programima Igra_kartama. Ruka i Spil klase su iste kao i u Igra_kartama2. Nova Karta klasa ima istu funkcionalnost kao i Pozicionisana_Karta iz Igra_kartama3.

```
# Cards Module
# Basic classes for a game with playing cards

class Card(object):
    """ A playing card. """
    RANKS = ["A", "2", "3", "4", "5", "6", "7",
             "8", "9", "10", "J", "Q", "K"]
    SUITS = ["c", "d", "h", "s"]

    def __init__(self, rank, suit, face_up = True):
        self.rank = rank
        self.suit = suit
        self.is_face_up = face_up

    def __str__(self):
        if self.is_face_up:
            rep = self.rank + self.suit
        else:
            rep = "XX"
        return rep

    def flip(self):
        self.is_face_up = not self.is_face_up

class Hand(object):
    """ A hand of playing cards. """
    def __init__(self):
        self.cards = []

    def __str__(self):
        if self.cards:
           rep = ""
           for card in self.cards:
```

```
                rep += str(card) + "\t"
        else:
            rep = "<empty>"
        return rep

    def clear(self):
        self.cards = []

    def add(self, card):
        self.cards.append(card)

    def give(self, card, other_hand):
        self.cards.remove(card)
        other_hand.add(card)

class Deck(Hand):
    """ A deck of playing cards. """
    def populate(self):
        for suit in Card.SUITS:
            for rank in Card.RANKS:
                self.add(Card(rank, suit))

    def shuffle(self):
        import random
        random.shuffle(self.cards)

    def deal(self, hands, per_hand = 1):
        for rounds in range(per_hand):
            for hand in hands:
                if self.cards:
                    top_card = self.cards[0]
                    self.give(top_card, hand)
                else:
                    print("Can't continue deal. Out of cards!")



if __name__ == "__main__":
    print("This is a module with classes for playing cards.")
    input("\n\nPress the enter key to exit.")

# Games
# Demonstrates module creation

class Player(object):
    """ A player for a game. """
    def __init__(self, name, score = 0):
        self.name = name
        self.score = score

    def __str__(self):
        rep = self.name + ":\t" + str(self.score)
        return rep

def ask_yes_no(question):
    """Ask a yes or no question."""
    response = None
    while response not in ("y", "n"):
        response = input(question).lower()
    return response

def ask_number(question, low, high):
```

```
    """Ask for a number within a range."""
    response = None
    while response not in range(low, high):
        response = int(input(question))
    return response


if __name__ == "__main__":
    print("You ran this module directly (and did not 'import' it).")
    input("\n\nPress the enter key to exit.")



# Blackjack
# From 1 to 7 players compete against a dealer

import cards, games

class BJ_Card(cards.Card):
    """ A Blackjack Card. """
    ACE_VALUE = 1

    @property
    def value(self):
        if self.is_face_up:
            v = BJ_Card.RANKS.index(self.rank) + 1
            if v > 10:
                v = 10
        else:
            v = None
        return v

class BJ_Deck(cards.Deck):
    """ A Blackjack Deck. """
    def populate(self):
        for suit in BJ_Card.SUITS:
            for rank in BJ_Card.RANKS:
                self.cards.append(BJ_Card(rank, suit))


class BJ_Hand(cards.Hand):
    """ A Blackjack Hand. """
    def __init__(self, name):
        super(BJ_Hand, self).__init__()
        self.name = name

    def __str__(self):
        rep = self.name + ":\t" + super(BJ_Hand, self).__str__()
        if self.total:
            rep += "(" + str(self.total) + ")"
        return rep

    @property
    def total(self):
        # if a card in the hand has value of None, then total is None
        for card in self.cards:
            if not card.value:
                return None

        # add up card values, treat each Ace as 1
        t = 0
        for card in self.cards:
```

```
                t += card.value

        # determine if hand contains an Ace
        contains_ace = False
        for card in self.cards:
            if card.value == BJ_Card.ACE_VALUE:
                contains_ace = True

        # if hand contains Ace and total is low enough, treat Ace as 11
        if contains_ace and t <= 11:
            # add only 10 since we've already added 1 for the Ace
            t += 10

        return t

    def is_busted(self):
        return self.total > 21


class BJ_Player(BJ_Hand):
    """ A Blackjack Player. """
    def is_hitting(self):
        response = games.ask_yes_no("\n" + self.name + ", do you want a hit? (Y/N): ")
        return response == "y"

    def bust(self):
        print(self.name, "busts.")
        self.lose()

    def lose(self):
        print(self.name, "loses.")

    def win(self):
        print(self.name, "wins.")

    def push(self):
        print(self.name, "pushes.")


class BJ_Dealer(BJ_Hand):
    """ A Blackjack Dealer. """
    def is_hitting(self):
        return self.total < 17

    def bust(self):
        print(self.name, "busts.")

    def flip_first_card(self):
        first_card = self.cards[0]
        first_card.flip()


class BJ_Game(object):
    """ A Blackjack Game. """
    def __init__(self, names):
        self.players = []
        for name in names:
            player = BJ_Player(name)
            self.players.append(player)

        self.dealer = BJ_Dealer("Dealer")
```

```
        self.deck = BJ_Deck()
        self.deck.populate()
        self.deck.shuffle()

    @property
    def still_playing(self):
        sp = []
        for player in self.players:
            if not player.is_busted():
                sp.append(player)
        return sp

    def __additional_cards(self, player):
        while not player.is_busted() and player.is_hitting():
            self.deck.deal([player])
            print(player)
            if player.is_busted():
                player.bust()

    def play(self):
        # deal initial 2 cards to everyone
        self.deck.deal(self.players + [self.dealer], per_hand = 2)
        self.dealer.flip_first_card()    # hide dealer's first card
        for player in self.players:
            print(player)
        print(self.dealer)

        # deal additional cards to players
        for player in self.players:
            self.__additional_cards(player)

        self.dealer.flip_first_card()    # reveal dealer's first

        if not self.still_playing:
            # since all players have busted, just show the dealer's hand
            print(self.dealer)
        else:
            # deal additional cards to dealer
            print(self.dealer)
            self.__additional_cards(self.dealer)

            if self.dealer.is_busted():
                # everyone still playing wins
                for player in self.still_playing:
                    player.win()
            else:
                # compare each player still playing to dealer
                for player in self.still_playing:
                    if player.total > self.dealer.total:
                        player.win()
                    elif player.total < self.dealer.total:
                        player.lose()
                    else:
                        player.push()

        # remove everyone's cards
        for player in self.players:
            player.clear()
        self.dealer.clear()


def main():
```

```
    print("\t\tWelcome to Blackjack!\n")

    names = []
    number = games.ask_number("How many players? (1 - 7): ", low = 1, high = 8)
    for i in range(number):
        name = input("Enter player name: ")
        names.append(name)
    print()

    game = BJ_Game(names)

    again = None
    while again != "n":
        game.play()
        again = games.ask_yes_no("\nDo you want to play again?: ")

main()
```

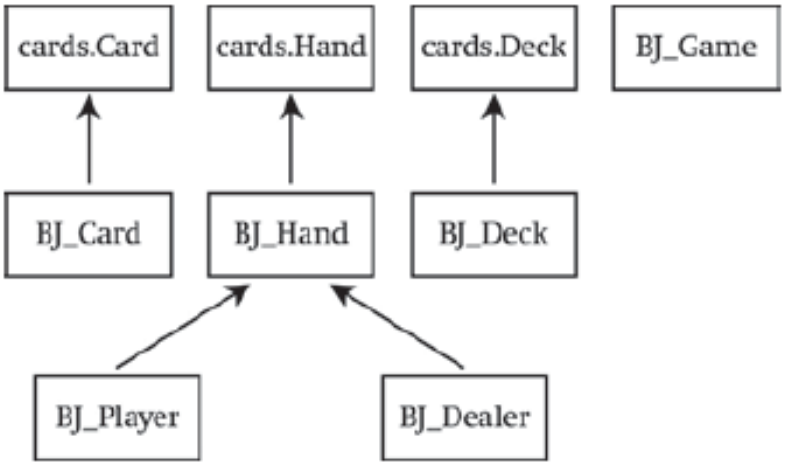
015 Dizajn klasa

Pre nego se započne kodovanje sa više klasa, može pomoći mapirati klase na papiru. Možda će pomoći pravljenje liste koja uključuje kratak opis svake klase. U tabeli je prva skica takve liste za igru Blackjack.

Klasa	Osnovna klasa	Opis
BJ_Card	cards.Card	Blekdžek karta za igranje. Definiše atribut value za prikaz vrednosti karte.
BJ_Deck	cards.Deck	Blekdžek špil. Kolekcija BJ_Card objekata.
BJ_Hand	cards.Hand	Blekdžek ruka. Definiše atribut total za prikaz sume vrednosti ruke. Definiše atribut name za prikaz vlasnika ruke.
BJ_Player	BJ_Hand	Blekdžek igrač.
BJ_Dealer	BJ_Hand	Blekdžek diler karata.
BJ_Game	object	Blekdžek igra. Definiše atribut deck za BJ_Deck objekat. Definiše atribut dealer za BJ_Dealer objekat. Definiše atribut players za listu od BJ_Player objekata.

Treba pokušati uključiti sve klase koje su potrebne. Ali pravljenje ovakve liste bi trebalo da da dobar prikaz tipova objekata sa kojima će se raditi u kodu.

Sledeći pomoćni crtež je porodično stablo koje pokazuje kako su klase međusobno povezane:



Ovo je dijagram klasne hijerarhije, i prikazuje kako se koristi nasleđivanje u programu.

016 Pisanje pseudokoda za petlju

U planiranju treba i napisati pseudokod za makar jednu rundu igre. Na taj način se bolje vidi kako objekti interaguju međusobno:

```
Deal each player and dealer initial 2 cards
For each player
        While the player asks for a hit and the player is not busted
                Deal the player an additional card
If there are no players still playing
        Show the dealer's 2 cards
Otherwise
        While the dealer must hit and the dealer is not busted
                Deal the dealer an additional card
        If the dealer is busted
                For each player who is still playing
                        The player wins
        Otherwise
                For each player who is still playing
                        If the player's total is greater than the dealer's total
                                The player wins
                        Otherwise, if the player's total is less than the dealer's total
                                The player loses
                        Otherwise
                                The player pushes
```

017 Importovanje modula cards i games

Posle planiranje kreće kucanje koda.

U prvom delu programa, importuju se moduli cards i games.

```
# Blackjack
# From 1 to 7 players compete against a dealer

import cards, games
```

Modul games je isti kao ranije korišćeni modul.

018 Klasa BJ_Card

Klasa BJ_Card proširuje definiciju šta je karta preko nasleđivanja od cards.Card. Unutar BJ_Card kreirano je novo svojstvo, value, za glavnu vrednost karte:

```
class BJ_Card(cards.Card):
    """ A Blackjack Card. """
    ACE_VALUE = 1

    @property
    def value(self):
        if self.is_face_up:
            v = BJ_Card.RANKS.index(self.rank) + 1
            if v > 10:
                v = 10
        else:
            v = None
        return v
```

Ova metoda vraća broj između 1 i 10, koji predstavlja vrednost blekdžek karte. Prvi deo računanja se završava preko izraza BJ_Card.RANKS.index(self.rank) + 1. Ovaj izraz uzima rank atribut iz ibjekta (npr “6“) i i pronalazi njegov odgovarajući broj indeksa unutar BJ_Card.RANKS

kroz list metodu index() (za “6“ to je 5). Na kraju, 1 se dodaje na rezultat pošto kompjuter počinje da broji od 0 (ovo čini da izračunata vrednost od “6“ bude korektna, 6). Ipak pošto rank atribut od “J“, “Q“ i “K“ daje broj veći od 10, bilo koja value veća od 10 je postavljena na 10. Ako je objektov face_up atribut False, ovaj ceo proces se preskače i vrednost None se vraća.

019 BJ_Deck klasa

Klasa BJ_Deck se koristi za kreiranje deka blekdžek karata. Klasa je skoro identična sa njenom osnovnom klasom, cards.Deck. Jedina razlika je ta što se prepravlja cards.Deck metoda populate() da bi se omogućilo popunjavanje novog BJ_Deck objekta sa BJ_Card objektima:

```
class BJ_Deck(cards.Deck):
    """ A Blackjack Deck. """
    def populate(self):
        for suit in BJ_Card.SUITS:
            for rank in BJ_Card.RANKS:
                self.cards.append(BJ_Card(rank, suit))
```

020 BJ_Hand klasa

Ova klasa, zasnovana na cards.Hand, se koristi za blekdžek špil u rukama igrača. Prepravljen je cards.Hand konstruktor i dodat name atribut za predstavljanje imena vlasnika špila u rukama:

```
class BJ_Hand(cards.Hand):
    """ A Blackjack Hand. """
    def __init__(self, name):
        super(BJ_Hand, self).__init__()
        self.name = name
```

Sledeće, prepravlja se nasleđena __str__() metoda za prikaz ukupnih bodova u rukama:

```
    def __str__(self):
        rep = self.name + ":\t" + super(BJ_Hand, self).__str__()
        if self.total:
            rep += "(" + str(self.total) + ")"
        return rep
```

Nadovezuje se objektov atribut name sa stringom koji se vraća iz cards.Hand__str__() metodom za objekat. Zatim, ako objektov total svojstvo nije None, nadovezuje se string vrednosti total, a zatim se vraća taj string.

Sledeće, kreira se svojstvo total, koje predstavlja ukupnu vrednost koja je u blakdžak ruci. Ako blekdžek ruka ima kartu okrenutu licem nadole, onda njeno total svojstvo je None. Inače, vrednost se izračunava dodavanjem vrednosti svih karata u ruci.

```
@property
    def total(self):
        # if a card in the hand has value of None, then total is None
        for card in self.cards:
            if not card.value:
                return None

        # add up card values, treat each Ace as 1
        t = 0
        for card in self.cards:
              t += card.value

        # determine if hand contains an Ace
        contains_ace = False
        for card in self.cards:
            if card.value == BJ_Card.ACE_VALUE:
                contains_ace = True
```

```
        # if hand contains Ace and total is low enough, treat Ace as 11
        if contains_ace and t <= 11:
            # add only 10 since we've already added 1 for the Ace
            t += 10

        return t
```

Prvi deo ove metode proverava da li bilo koja karta u blekdžek ruci ima vrednost identičnu sa None (što bi trebalo da znači da je karta okrenuta licem nadole). Ako je tako, metod vraća None. Sledeći deo metode samo sumira vrednsoti od svih karata u ruci. Sledeći deo odlučuje ako ruka sadrži as. Ako je tako, poslednji deo metode određuje da li je vrednost karte 11 ili 1. Poslednja metoda u BJ_Hand je is_busted(). Ona vraća True ako je objektov total svojstvo veće od 21, inače vraća False.

```
def is_busted(self):
        return self.total > 21
```

U ovoj metodi, vraća se rezultat uslova self.total > 21 umesto da se dodeljuje rezultat nekoj promenjivoj a zatim da se vraća ta promenjiva. Može se kreirati ovakva vrsta return iskaza sa bilo kakvim uslovom (ili bilo kakvim izrazom) i često rezultuje kao elegantniji metod rada.

Ovakva vrsta metode, koja vraća ili True ili False, se često koristi. Često se koristi za prikaz uslova objekata sa dve mogućnosti (on ili off). Ova vrsta metode najčešće ima naziv koji počinje sa is, poput is_on().

021 BJ_Player klasa

Klasa BJ_Player je izvedena iz BJ_Hand, i koristi se za blekdžek igrače:

```
class BJ_Player(BJ_Hand):
    """ A Blackjack Player. """
    def is_hitting(self):
        response = games.ask_yes_no("\n" + self.name + ", do you want a hit? (Y/N): ")
        return response == "y"

    def bust(self):
        print(self.name, "busts.")
        self.lose()

    def lose(self):
        print(self.name, "loses.")

    def win(self):
        print(self.name, "wins.")

    def push(self):
        print(self.name, "pushes.")
```

Prva metoda, is_hitting(), vraća True ako igrač želi sledeći hit i vraća Flase ako ne želi. Metoda bust() najavljuje da je igrač propao i poziva objektovu lose() metodu. Metoda lose() najavljuje da je igrač izgubio. Metoda win() najavljuje da igrač pobeđuje. Metoda push() najavljuje da igrač ide dalje. Sve ove metode su jednostavne pa s emože postaviti pitanje zašto uopšte i postoje. Smeštene su u klasu zato što formiraju odličan skelet strukture koja može podržati kompleksnije probleme koji se pojavljuju kada je igraču dozvoljeno da se kladi.

022 BJ_Dealer klasa

Klasa BJ_Dealer, izvedena iz BJ_Hand, se koristi za blekdžek dilera:

```
class BJ_Dealer(BJ_Hand):
    """ A Blackjack Dealer. """
    def is_hitting(self):
        return self.total < 17
```

```
    def bust(self):
        print(self.name, "busts.")

    def flip_first_card(self):
        first_card = self.cards[0]
        first_card.flip()
```

Prva metoda, is_hitting(), predstavlja da li diler uzima ili ne dodatne karte. Pošto diler mora uraditi hit na bilo koju ruku koja ima 17 ili manje, metoda vraća True ako je objektov total osobenost manja od 17; inače, vraća False. Metoda bust() najavljuje da je diler propao. Metoda flip_first_card() vraća dilerovu prvu kartu.

023 BJ_Game klasa

Klasa BJ_Game se koristi za kreiranje jednog objekta koji predstavlja blekdžek igru. Klasa sadrži kod za petlju glavne igre, unutar play() metode. Ipak, mehanika igre je kompleksna pa su kreirani nekoliko elemenata izvan metode, uključujući __additional_cards() metoda koja vodi računa o deljenju dodatnih karata igraču a still_playing osobenost koja vraća listu svih igrača koji još uvek igraju u rundi.

- __init__() metoda

Konstruktor dobija listu imena i kreira igrača za svako ime. Metoda takođe kreira dilera i dek.

- still_playing osobenost

Ova osobenost vraća listu svih igrača koji su još uvek u igri (oni koji nisu propali u tekućoj rundi).

- __aditional_cards metoda

Ova metoda dodaje dodatne karte igraču ili dileru. Metoda dobija objekat u svoj player parametru, koji može biti ili BJ_Player ili BJ_Dealer objekat. Metoda nastavlja dok objektova is_busted() metoda vraća False a is_hitting() metoda vraća True. Ako objektova is_busted() metoda vrati True, onda se poziva bust() metoda. Ovde se koristi polimorfizam u pozivu dve metode. Metoda player.is_hitting() poziva bez obzira da li se player odnosi na BJ_Player objekat ili na BJ_Dealer objekat. Metoda __additional_cards() nikada ne mora znati sa kojim tipom objekta radi. Isto je tačno sa player.bust(). Pošto obe klase, BJ_Player i BJ_Dealer, obe definišu svoje bust() metode, linija kreira željeni rezultat u oba slučaja.

- play() metoda

Ova metoda se nalazi gde se definiše petlja igre i ima sličnost sa prethodno napisanim pseudokodom. Svaki igrač i diler dobija inicijalno dve karte. Dilerova prva karta se okrene da bi sakrila vrednost. Zatim se sve karte u rukama prikažu. Svaki igrač dobije karte po zahtevu i ako nije propao. Ako su svi igrači propali, dilerova prva karta se okreće i njegova ruka se prikazuje. Inače, igra se nastavlja. Diler dobija karte sve dok njegova ruka je manja od 17. Ako je diler propao, svi ostali igrači pobeđuju. Inače, karte u rukama svakog od preostalih igrača se upoređuje sa dilerovom rukom. Ako je igračev skor veći od dilerovog, pobeđuje igrač. Ako je igračeva suma manja, igrač gubi. Ako su dve sume iste, igrač nastavlja.

024 main() funkcija

Funkcija main() dobija imena svih igrača, stavlja ih na listu, kreira BJ_Game objekat koristeći listu kao argument. Sledeće, funkcija poziva objektov play() metodu i nastavlja će to da radi sve dok igrači žele da igraju.