Potrebno je napisati dvije klase: `InvalidIPv6AddressException` i `IPv6Utils`.

Klasa `InvalidIPv6AddressException` mora nasljeđivati provjeravanu iznimku i sadržavati konstruktor u kojem će se primljena poruka tipa String prosljeđivati nadklasi.

U klasi `IPv6Utils` potrebno je napisati statičku metodu `checkIPv6AddressValidity` koja prima jedan podatak tipa String i na vraća ništa (`void`). U toj metodi je potrebno implementirati jednostavnu provjeru ispravnosti primljene IPv6 adrese te, u slučaju da primljena IPv6 adresa nije u ispravnom formatu, baciti iznimku `InvalidIPv6AddressException` s odgovarajućom porukom (3 tipa poruke su nabrojana ispod).

**Ispravna IPv6 adresa** je u formatu **x:x:x:x:x:x:x:x** gdje je x heksadekadski broj u rasponu od 0 do FFFF. Primjeri ispravnih IPv6 adresa:

- **2001:0db8:0001:0000:FFFF:0ab9:C0A8:0102**
- **2001:db8:3333:4444:5555:6666:7777:8888**

Ako je **IPv6 adresa prekratka ili predugačka**, tj. ne sadrži 8 heksadekadskih brojeva odvojenih dvotočkom (npr. **2001:0db8:0001:** ili **2001:0db8:0001:0000:FFFF:0ab9:C0A8:FFFF:0202**), metoda treba baciti iznimku `InvalidIPv6AddressException` s porukom `IPv6 address does not contain 8 hexadecimal numbers!`.

Ako **IPv6 adresa sadrži znakove koji nisu heksadekadski** (npr. **2001:0db8:!!!!:XXXX:FFFF:0ab9:C0A8:0102**), metoda treba baciti iznimku `InvalidIPv6AddressException` s porukom `One or more strings in the IPv6 address are not hexadecimal numbers!`.

Ako **IPv6 adresa sadrži heksadekadske brojeve koji nisu u rasponu od 0 do FFFF** (npr. **2001:0db8:0001:0000:FFFFF:0ab9:C0A8:FFFFF**), metoda treba baciti iznimku `InvalidIPv6AddressException` s porukom `One or more numbers in the IPv6 address are not in the correct range!`.

Nabrojani uvjeti se moraju provjeravati redom, tj. ako metoda primi ulaz **2001:XXXX:FFFFF**, treba baciti iznimku s porukom `IPv6 address contains fewer than 8 hexadecimal numbers!`, jako niz sadrži znakove koji nisu heksadekadski i heksadekadski broj koji je izvan traženog raspona.

Imamo apstraktnu klasu koja predstavlja kupovni artikl:

```
abstract class Artikl<T>{
    private T tag;
    private double pricing;
    private Type type;

    public Artikl(T tag, double pricing, Type type) {
        this.tag = tag;
        this.pricing = pricing;
        this.type = type;
    }

    public T getTag() {
        return tag;
    }

    public double getPricing() {
        return pricing;
    }
```

```
    public Type getType() {
        return type;
    }

    public abstract void setQuantity(double quantity);

    public abstract double getPrice();
}
```

Svaki artikl ima člansku varijablu `tag` koja može biti ili ime artikla (`String`) ili identifikacijski broj (id) artikla (`Integer`). Imamo dvije vrste artikala: one koje kupujemo po komadu (lubenica, televizor) i one koje kupujemo po kilogramu (jabuke, bademi). Enumeracijom `Type` opisujemo ta dva tipa proizvoda --> `ITEM` po komadu i `KG` po kilogramu:

```
enum Type { ITEM, KG }
```

Artikl također ima varijablu pricing koja predstavlja cijenu artikla po komadu ili po kilogramu artikla ovisno o tipu artikla.

***Potrebno je napisati klasu*** `Items` koja nasljeđuje apstraktnu klasu Artikl i ima dodatnu člansku varijablu koja nam govori koliko imamo ukupno artikla `private double quantity`.

U slučaju kad se proizvod naplaćuje po komadu, `quantity` ne smije imati decimale.

U slučaju kad se proizvod naplaćuje po kilogramu, `quantity` označuje količinu proizvoda *** U GRAMIMA *** i smije imati decimale.

Varijabla `quantity` ne smije biti manja od 0.

Metoda `setQuantity(double quantity)` postavlja vrijednost varijable `quantity` s tim da treba izbaciti `IllegalArgumentException("For pricing per item, the quantity cannot have decimals")` u slučaju kada pokušamo postaviti količinu s decimalama na proizvod koji se prodaje po komadu. Npr:

```
Items<String> lubenica = new Items<String>("lubenica", 10, Type.ITEM);
lubenica.setQuantity(3.2);   // --> IllegalArgumentException
```

Također, potrebno je izbaciti `IllegalArgumentException("Negative values forbidden")` prilikom pokušaja postavljanja negativne količine proizvoda.

Metoda `getPrice()` vraća ukupnu cijenu proizvoda. Npr:

```
Items<Integer> jabukaCrvena = new Items<Integer>(123, 80, Pricing.KG);
jabukaCrvena.setQuantity(500.5);
System.out.println(jabukaCrvena.getPrice()) --> Ispisuje 40.04
```

Ako varijabla `quantity` nije postavljena, automatski se postavlja na 0.

**Napomena:** Klasu *Items* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripadaju. Na vrhu možete dodavati *importe* iz standardnih javinih biblioteka.

Napisati klasu *KeyValue* parametriziranu po 2 parametra koja predstavlja parove ključ vrijednost. Za klasu *KeyValue* definirati *gettere* i *settere* za ključ i vrijednost te konstruktor koji prima inicijalne vrijednosti ključa i vrijednosti. Npr. sljedećim retkom

```
KeyValue<String, Integer> kv = new KeyValue<>("pero", 5);
```

stvorit će se novi objekt u kojem će ključ biti `pero`, a za vrijednost `5`.

Nakon toga iz klase *KeyValue* izvesti klasu *SingleTypeKeyValue* koja predstavlja parametrizirani par ključ-vrijednost u kojem su ključ i vrijednost istog tipa. Klasa *SingleTypeKeyValue* ne definirane nikakve dodatne metode u odnosu na *KeyValue*.

Nakon toga iz klase *SingleTypeKeyValue* izvesti klasu *XY* koja predstavlja par vrijednosti tipa *Double*. Smisao ove klase je da predstavlja vrijednost neke funkcije za neku vrijednost x koja je zapisana kao ključ.

U klasi *XY* napisati statičku metodu `public static boolean isIncreasingFunction(XY[] data)` koja za primljeno polje provjerava jesu li parovi koji predstavljaju parove (x,y) neke funkcije takvi da je funkcija rastuća. Poredak elemenata u polju može biti proizvoljan i **polje se ne smije mijenjati**.

*Primjer: Ako je ulazno polje sadržavalo sljedeće vrijednoti za XY* `(6, 8), (7, 8), (9.1, 13), (-3, -2), (-1, 4)`*, funkcija je rastuća (primijetiti da ne mora biti stroga rastuća), a npr. za polje sadržaje* `(6, 8), (7.5, 12), (9, 9), (10, 10), (11, 11)` *ukazuje da funkcija nije rastuća.*

Ako je u metodu *isIncreasingFunction* poslano prazno polje ili polje sa samo jednim elementom, metoda mora vratiti *true*. Možete pretpostaviti da su vrijednost za x ispravno zadane (tj. ne može se dvaput pojaviti ista vrijednost za x).

**Napomena:** Klase *KeyValue*, *SingleTypeKeyValue* i *XY* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na vrhu možete dodavati *importe* iz standardnih Javinih biblioteka.

U sustavu postoje sljedeće klase (kao u primjeru s predavanja):

**Item, Food, Cloth, Beverage, Milk**

te dodatno klasa **AlcBeverage** koja nasljeđuje **Beverage** i uvodi jednu novu varijablu (postotak alkohola, *alc*):

```
public class AlcBeverage extends Beverage {
    private double alc;

    public AlcBeverage(String sku, String name, double price, double volume, double alc) {
        super(sku, name, price, volume);
        this.alc = alc;
    }

    public String getItemType() {
        return "AlcBeverage";
    }

    public double getAlc() {
        return alc;
    }
```

```
public void setAlc(double alc) {
    this.alc = alc;
}
```

Naspisati klasu `IllegalAgeException` koja nasljeđuje **RuntimeException** i sadrži konstruktor kojim se može postaviti poruke iznimke.

Napisati klasu `MaliExpres` i u njoj metodu `public static double checkout(ShoppingCart cart);` koja će koristeći parametar `ShoppingCart cart` postojeće klase **ShoppingCart** dohvatiti polje objekata tipa Item (artikl) i vratiti ukupnu cijenu svih artikala. Cijena pojedinog artikla može se dobiti preko virtualne metode **public double getPrice(int count)**, u klasi **Item**, gdje je count=1.

Klasa **ShoppingCart** ima virtualne metode:

```
public Item[] getItems(); //vraća referencu na polje svih artikala u košarici.
public Customer getCustomer(); //vraća referencu na kupca, vlasnika košarice.
```

Ako je neki od artikala čiju cijenu trebate dodati u ukupnu sumu košarice alkoholno piće (tj. tipa AlcoholicBeverage) tada morate provjeriti koliko godina ima kupac. To se radi pozivom metode `public int getAge();` iz klase **Customer**. Ako kupac ima manje od 18 godina, vaša metoda mora baciti **IllegalAgeException** s porukom `Nije dozvoljena prodaja alkohola maloljetnicima`.

Uočite da bi se i pri **dohvaćanju artikala iz košarice** mogla dogoditi iznimka (npr. Null pointer exception - Artikl je obrisan iz košarice ali je greškom u listi ostala null referenca) - u tom slučaju iznimku treba "obraditi" tako da se taj artikl zanemari, a čitanje i obrada se nastavlja dalje sa **sljedećim artiklom**)

Primjer 1: neka se u košarici nalaze dva artikla (Kruh tipa Food s cijenom 7kn, i Cedevita tipa Beverage s cijenom 12kn). Metoda *checkout* treba vratiti **19.00**.

Primjer 2: neka se u košarici nalaze tri "artikla" (Kruh tipa Food s cijenom od 7kn, null referenca, i Pivo tipa AlcBeverage s cijenom od 15kn). Metoda *checkout* treba pribrojiti prvu cijenu u sumu, pa progutati iznimku, i onda provjeriti dob kupca zbog trećeg artikla. Ako kupac ima manje od 18 godina, baciti

**IllegalAgeException** iznimku s odgovarajućom porukom. U protivnom metoda *checkout* treba u ukupnu sumu pribrojiti i cijenu tog artikla, pa će vratiti **22.00**.

**Napomena:** Klase *IllegalAgeException* i *MaliExpres* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripadaju. Klase i sučelja iz 5. predavanja (Item, Food, Cloth, Beverage, Perishable, Milk te klasa AlcBeverage) su zadane i njih **NE predajete** s rješenjem. Na vrhu možete dodavati *importe* iz standardnih javnih biblioteka.

Napisati klasu `Utils` i u njoj javnu statičku metodu `checkPassword` koja prima jedan podatak tipa String i vraća cijeli broj koji predstavlja ocjenu kompleksnosti provjerene lozinke, i to na sljedeći način:

** ocjena = (duljina lozinke) + (broj znamenki)*3 + (broj velikih slova)*2 **

Npr. za lozinku `1234aBcd` funkcija treba vratiti `22` (8 + 4*3 + 1*2)

U slučaju da je unesena lozinka kraća od 6 znakova funkcija treba **baciti** iznimku tipa **IllegalArgumentException**, a ako je ocjena lozinke manja od 10, funckija treba baciti **SecurityException**. Ako se poklope oba uvjeta, funkcija **baca IllegalArgumentException**.

**Napomena:** Metoda *checkPassword* mora biti javna i statička. Klasu *Utils* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripada. Na vrhu možete dodavati importe iz standardnih Javinih biblioteka.

Napisati parametriziranu klasu Triple koja predstavlja trojku nekog tipa (npr. trojka *Stringova*, trojka *Integera*, itd.). Za klasu Triple definirati:

- metodu *getElement* koja prima cijeli broj (1,2 ili 3) te vraća odgovarajući član trojke.
- metodu setElement koja prima dva argumenta: cijeli broj (1, 2 ili 3) i vrijednost koju treba postaviti kao člana trojke na odgovarajuću poziciju u trojci.
    - U obje metode u slučaju neispravnog argumenta baciti iznimku tipa *IllegalArgumentException*
- konstruktor s 3 argumenta kojim se postavljuju inicijalne vrijednosti trojke

Nakon toga treba napisati klasu *Vector3D* koja predstavlja trojku čiji su članovi brojevi (koristiti tip *Number*) te u klasi Vector3D napisati statičku metodu koja će izračunati skalarni produkt dviju trojki. Skalarni produkt se računa kao suma produkata članova na istim mjestima u trojci.

Primjer nekih naredbi u glavnom programu:

```
double x = Vector3D.dotProduct(new Vector3D(-1, 2, 5), new Vector3D(4, -3, 7));
//x je 25.0 zbog -1 * 4 + 2 * -3 + 5 * 7
//
Triple<Number>  t1 = new Vector3D(-1, 2.3, 5);
```

**Napomena:** Klase *Triple* i *Vector3D* napisati bez modifikatora vidljivosti i kopirati u prostor za rješenje bez navođenja paketa kojem pripadaju. Na vrhu možete dodavati proizvoljne *importe*.