Házi feladat megoldások:

Példa:

Az előző két példában megismertek alapján, tároljanak tetszőleges mennyiségű nevet rendezett láncolt listában.

Megbeszélés:

1. Beszúrásos rendezés.

Megoldás:

```
package láncolt_lista_rendezett;
import static java.lang.System.out;
import java.util.LinkedList;
import java.util.Scanner;
public class Láncolt_Lista_Rendezett {
	public static void main(String[] args) {
	int i;
	Scanner billentyu = new Scanner (System.in);
	LinkedList<String> nevek = new LinkedList<>();
	out.print("Kérek egy nevet: ");
	String nev = billentyu.nextLine();
	while(nev.length()!=0){
		for(i=0; i<nevek.size(); i++){
			if(nev.compareTo(nevek.get(i)) < 0){
				nevek.add(i, nev);
				break;
			}
		}
		if( i == nevek.size())nevek.add(nev);
		out.print("Kérek egy nevet: ");
		nev = billentyu.nextLine();
	}
	out.println(nevek);
}}
```

Példa:

Készítsünk programot, amely billentyűzetről feltölti az alábbi összetett adatszerkezetet (struktúrát), majd *azon*-ra rendezetten altárolja egy láncolt listában. Az adatokat addig olvassa be a program, amig az *azon*-ra nullát nem adunk. A feladatot beszúró rendezéssel oldják meg.

int azon;
string név;
float fizetés;

Megoldás:

```
package láncolt_lista_rendezett_ob;
import static java.lang.System.out;
import java.util.LinkedList;
import java.util.Scanner;
public class Láncolt_lista_rendezett_OB {
        public static void main(String[] args) {
                int i;
                Scanner bill = new Scanner (System.in);
                LinkedList<OB_def> adatok = new LinkedList<>();
                OB_def OB = new OB_def();
        //Rendezett feltöltés
                out.printf("Kérem az azonosítót: ");
                OB.azon = bill.nextInt();
                while(OB.azon!=0){
                    out.printf("Kérem a nevet: ");
                    OB.nev = bill.next();
                    out.printf("Kérem a fizetést: ");
                    OB.fizetes = bill.nextFloat();
                    for(i=0; i<adatok.size(); i++){
                            if(adatok.get(i).azon > OB.azon){
                                adatok.add(i, OB);
                                break;
                    }       }
                    if( i == adatok.size())adatok.add(OB);
                    OB = new OB_def();
                    out.printf("Kérem az azonosítót: ");
                    OB.azon = bill.nextInt();
                }
        //Kiírás
                for(i = 0; i < adatok.size(); i++)
          out.printf("%10d %20s %10.2f\n",adatok.get(i).azon, adatok.get(i).nev, adatok.get(i).fizetes);
          }
          public static class OB_def{
                public Integer azon;
                public String nev;
                public Float fizetes;
          }
        }
```

Hibakezelés (kivételkezelés)

A kivételkezelés célja, hogy a program futása során keletkezett hibákat kiszűrjük és megfelelő módon le tudjuk kezelni. Két különböző kivétel (Exception) létezik: a futási időben és a nem futási időben keletkezett kivételek. A futásidejű kivételeket (pl. nullával való osztás, túlindexelés) nem kötelező lekezelni. A nem futásidejű kivételek a Java rendszerén kívül keletkeznek (pl. I/O műveletek), amelyeket kötelező lekezelni.
Ha egy metódus végrehajtása közben kivétel keletkezik, két dolgot tehetünk. Az egyik, ha nem akarjuk, vagy nem tudjuk a metóduson belül lekezelni hibát, akkor azt tovább kell küldenünk a hívó metódusnak. Amennyiben a keletkezett kivételt le akarjuk kezelni, akkor ehhez el kell készítenünk a megfelelő programrészeket.

1. Nem kezeljük le a hibát

   Ezt a metódus fejlécébe írt *throws* kulcsszóval tehetjük meg, utána felsorolva azokat a kivételosztályokat, amelyeket nem kezelünk le. Például:

   ```
   public static void main(String[] args) throws IOException
   ```

   Példa:

   Olvassunk be egy karaktert, ha szám, akkor írjuk ki szöveggel, egyébként a „Nem szám” szöveget.

   Megbeszélés:

   1. Karakter beolvasására a *Scanner* osztályt nem tudjuk használni.
   2. Az *io* csomag használata. Az *io* csomag osztályait csak a kivételek kezelésével használhatjuk, vagy eldobjuk a kivételeket. Az eldobáshoz a metódus fejrészének a végére a *throws IOException* kifejezést írjuk.

   Megoldás:

```
package switch_5_3;
import static java.lang.System.out;
import static java.lang.System.in;
import java.io.IOException;
public class Switch_5_3 {
        public static void main(String[] args) throws IOException{
                out.print("Kérek egy tetszőleges karaktert: ");
                char karakter = (char) in.read();
                switch (karakter){
                        case '0': out.println("nulla"); break;
                        case '1': out.println("egy"); break;
                        case '2': out.println("kettő"); break;
                        case '3': out.println("három"); break;
                        case '4': out.println("négy"); break;
                        case '5': out.println("öt"); break;
                        case '6': out.println("hat"); break;
                        case '7': out.println("hét"); break;
                        case '8': out.println("nyolc"); break;
                        case '9': out.println("kilenc"); break;
                        default: out.println("Nem szám karakter"); break;
        }       }
```

Példa:

Készítsünk egy szállodai vendégnyilvántartó programot. A szobák száma a tömb indexe legyen, a tagjai: bérlő neve, szobatársak száma és szobaár.
Használjuk a magyar ékezetes betűket!

Megbeszélés:

1. Osztály ⇒ objektum összeállítása.
2. Javában használatos karakterkódod-táblák.
3. Magyar ékezetes betűk használata.
4. Számformátum, *NumberFormat* osztály.

Megoldás:

```
package objektum_tömb_1;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import static java.lang.System.out;
import java.util.Scanner;
import java.text.NumberFormat;
public class Objektum_tömb_1 {
	public static void main(String[] args) throws UnsupportedEncodingException {
		PrintStream out = new PrintStream(System.out, true, "UTF-8");

		Szoba_OB szoba[] = new Szoba_OB[5];
		Scanner bill = new Scanner(System.in, "ISO8859_2");
//Tömb feltöltés
		for(int szobaszam = 0; szobaszam < 1; szobaszam++) {
			szoba[szobaszam] = new Szoba_OB();
			szoba[szobaszam].Be(bill);
		}
//Tömb kiírása
		out.println("Szobaszám  Vendég neve  Társak száma\tÁr");
		for(int i = 0; i<1; i++){
			out.printf("\t%d\t", i+1);
			szoba[i].Ki();
		}
	}
}
class Szoba_OB{
	private String vendeg;
	private int tars;
	private double ar;
	final NumberFormat penznem = NumberFormat.getCurrencyInstance();
	public void Be(Scanner bill) {
		out.print("Kérem a vendég nevét: ");
		vendeg = bill.next();
		out.print("Kérem a társak számát: ");
		tars = bill.nextInt();
		out.print("Kérem a szoba árát: ");
		ar = bill.nextDouble();
```

```
        }
        public void Ki() {
                out.print(vendeg);
                out.print("\t\t");
                out.print(tars);
                out.print("\t");
                out.println(penznem.format(ar));
        }
}
```

2. Lekezeljük le a hibát

Ebben az esetben azokat a programrészeket, amelyek a hibákat okozhatják, egy úgynevezett *try* blokkba tesszük:

```
try {
 // Megkísérelünk végrehajtani olyan utasításokat,
// amelyek kivételt okozhatnak.
}
catch(Exception error) {
// Ha kivétel történik, a program ide ugrik,
// itt megadhatjuk, hogy hiba esetén mi történjen.
// lehet többszörös, ekkor az általános hibakezelőt a végére kell tenni.
}
finally {
 // Nem kötelező megadni.
// Erre az ágra mindkét esetben (try, catch) átadódik a vezérlés.
// Ide kerülnek azok az utasítások, amelyeket mindenképpen végre
// kell hajtani (pl. fájllezárás).
// Mivel itt is lehet olyan művelet (pl. fájllezárás), ami szintén kivételt
// eredményezhet, ezt is illik try-catch szerkezettel lekezelni.
}
```

A try-catch-finally sorrendje rögzített!

Feladat:

Írjunk programot, amely lekezeli ha a nextInt() metódus használatakor nem egész számot adunk meg.

Olvassunk be egy tömbbe tetszőleges mennyiségű egész számot. A beolvasást egy nem egészszám karakterrel fejezzük be. Használjuk a kivételkezelést!

Megoldás:

```
package kivét_1;
import static java.lang.System.out;
import java.util.Scanner;
public class Kivét_1 {
        public static void main(String[] args) {
                Scanner billentyu = new Scanner (System.in);
                int[] tomb = new int[100];
                int db = 0;
```

```
        // Feltöltés
                do{
                    out.printf("Kérem a %d. számot: ",db+1);
                    try{
                        tomb[db] = billentyu.nextInt();
                    }
                    catch (Exception error){
                        break;
                    }
                    db++;
                }while(true);
        // Kiírás
                for(int i=0; i<db ;i++) out.println(tomb[i]);
        }       }
```

Feladat:

Írjunk programot, amely egy áru egységárából és a raktározott készletből kiszámítja az összértéket. A keletkező hibákat kezeljük le, Írjunk saját kivételtípust.

Megoldás:

```
package kivét_2;
import static java.lang.System.out;
import static java.lang.System.err;
import java.text.NumberFormat;
import java.util.Scanner;
public class Kivét_2 {
    public static void main(String[] args) {
        final double egységár = 1000;
        Scanner bill = new Scanner(System.in);
        NumberFormat pénz = NumberFormat.getCurrencyInstance();
        out.print("Áruk mennyisége: ");
        String áruk = bill.next();
        try{
            int db = Integer.parseInt(áruk);
            if(db < 0)throw new HibásÉrték();
            out.print("Az érték ");
            out.println(pénz.format(db * egységár));
        }
        catch (NumberFormatException error){
            err.println ("Nem számot adtunk meg! - " + error.getMessage());");
        }
        catch (HibásÉrték error){
            err.println("Hibás az áruk száma!");
        }
        finally{
            out.println("Ez a finally rész");
}       }       }
class HibásÉrték extends Exception{}
```

Feladat:

Írjuk át a korábban megírt jelszó ellenőrző programot, az ellenőrzést metódus végezze. Használjunk kivételkezelést.

Megoldás:

```
package kivétel_3;
import static java.lang.System.out;
import java.util.Scanner;
public class Kivétel_3 {
	public static void main(String[] args) {
		Scanner billentyu = new Scanner (System.in);
		out.print("Mi a jelszó? ");
		String jelszo = billentyu.next();
		out.println(teszt(jelszo));
	}
	public static String teszt(String jelszo){
		try{
			if(jelszo.equals("alma")) return"A jelszó rendben van";
			else throw new Exception();
		}
		catch(Exception err){
			return "A beírt jelszó hibás!";
		}
	}
}
```

Hallgatói feladatok:

1. Készítsen programot, amelyben egy listát (dinamikos tömböt) feltölt tetszőleges számú elemmel. Kezelje le, ha lekérdezésnél (*lista.get(szam)*) a szám nagyobb, mint az elemek száma. Írjon ki a hibának megfelelő hibaüzenetet. Használja a try, catch és finally kulcsszavakat.
2. Készítsen programot, amely egy egész típusú változó túlcsordulása esetén keletkező hibát lekezeli. Írjon ki a hibának megfelelő hibaüzenetet. Használja a try, catch és finally kulcsszavakat.
3. Készítsen programot, amely kiszámítja egy szám négyzetgyökét *(Math.sqrt())*. Írjon ki hibaüzenetet, ha a gyökvonás eredménytelen (pl. negatív számot adtunk meg) alkalmazza a *Double.isNaN()* metódust. Használja a try, catch és finally kulcsszavakat.
4. Készítsen programot, amelyben egy sztringbe beolvasott adatot, átkonvertál egésszé. Ha nem szám karaktert adtunk meg, akkor írjon ki a hibának megfelelő hibaüzenetet. Használja a try, catch és finally kulcsszavakat.
5. Készítsen programot, amely feltölt egy tömböt egész számokkal. A feltöltés során ellenőrzi, hogy a megadott szám szerepel-e a tömbben, ha igen, akkor adjon hibajelzést. A hibafigyelést kivételkezeléssel oldja meg.
6. Készítsen programot, amely feltölt egy sztringeket tartalmazó listát szöveggel. A feltöltés során ellenőrzi, hogy a megadott szám szerepel-e a listában, ha igen, akkor adjon hibajelzést. A hibafigyelést kivételkezeléssel oldja meg.