1. A java kód fordításának lépései

* Megírjuk a programkódot egy .java kiterjesztésű szöveges fájlba, majd ezt a javac compilert használva egy .class fájlba fordítjuk át.
* A .class fájlunk a Java Virtual Machine (JVM) számára értelmezhető bytekódokat tartalmaz, a java launcher ezt használva indítja el a programunkat
* A Java Development Kit biztosítja a szükséges fejlesztési eszközöket
* A Java Runtime Environment segít betölteni ezeket a JVM számára, hogy futtatni tudja

1. JVM: Mi az, miért fontos?

* Java Virtual Machine
* Ezen futnak a java kódok, a java platform része, ettől platformfüggetlen a nyelv

1. Mire lehet használni: JRE, JDK, IDE?
   * JRE: Java Runtime Environment – kapcsolatot biztosít a java és az operációs rendszer között (a memóriához való hozzáférés, IO műveletek)
   * JDK: Java Development Kit – a szükséges eszközöket biztosítja a java fejlesztéshez, javac compiler például
   * IDE: Integrated Development Environment – szövegszerkesztő, ami rengeteg hasznos funkcióval rendelkezik a kódunk megírásához, a szoftverfejlesztés folyamatát gyorsítja (Eclipse, IntelliJ)
2. Main metódus: hogy néz ki, miért fontos, mi a szerepe?
   * public static void main(String[] args)
   * Ez a belépési pontunk, kötelező tartalmaznia a programunknak
   * Paraméterként egy String tömböt vár el, ami a parancssori argumentumok listája
3. Sorolja fel és jellemezze a primitív adattípusokat és a wrapper osztályaikat!
   * Adatokat tudunk velük tárolni, különböző típusai vannak
   * Egész számokat tárol – int, byte, long, short
   * Tört számokat tárol – double, float
   * Logikai értéket tárol – boolean
   * Karaktereket tárol – char
   * A wrapper osztályok a primitívek változatai, rengeteg extra eljárást tartalmaznak, nagybetűvel kezdődik a nevük, objektumoknak számítanak, nem primitíveknek
4. String osztály
   * A String egy karakter tömb
   * Idézőjelek között lehet megadni („”)
   * Objektumnak számít
   * Deklaráláskor új objektumot készít (String nev = „hello”);
5. Tömb (array) deklarációja, használata
   * A tömn N darab érték lefoglalása egymás mellett
   * Deklaráláskor az adattípus mögé négyzetes zárójelet ([]) írunk, utána new adattípus[N]-ként tudjuk lefoglalni a memóriában
   * int[] valtozo = new int[6];
   * valtozo[N]-ként tudunk referálni arra, hogy hanyadik elemet szeretnénk
6. Általánosan az objektum és osztály definíciója
   * Osztály: olyan sablon/minta, amely primitíveket, objektumokat, metódusokat tartalmaz, amik leírják az objektumok viselkedését
   * Objektum: egy osztálynak a példánya
7. Absztakció fogalma
   * A modellezésnél fontos tulajdonságok, viselkedések leírása
8. Egységbezárás
   * Szabályozza, hogy egy objektum egyes elemeit miként érhetjük el
   * El tudunk vele rejteni adattagokat, függvényeket, eljárásokat
9. Mi a különbség az objektum állapota és viselkedése között?
   * Állapota alatt az attribútumait értjük
   * Viselkedése alatt a metódusait, függvényeit értjük
10. Mi az osztályattribútum és osztálymetódus?
    * Az objektum állapota és viselkedése helyett osztályszinten tudjuk vizsgálni az attribútumokat és metódusokat
11. Mi az a getter és setter?
    * Adatelrejtésre használjuk, hogy ne lehessen közvetlen módosítani az osztály attribútumait
    * Getter: az értékek eléréséhez használjuk
    * Setter: az értékek változtatására használjuk
12. Mi az öröklés (általánosan)?
    * Az öröklés olyan kapcsolattípus, ahol lehetővé tesszük az osztályok tulajdonságainak és metódusnak öröklődését
13. Mi az aggregáció?
    * Egy osztály egy másik osztály attribútumait tartalmazza
    * Kapcsolatuk laza, nem feltétlen kölcsönös
14. Mi a kompozíció?
    * Egy osztály egy másik osztály attribútumait tartalmazza
    * Kapcsolatuk szoros
15. Mi az asszociáció?
    * Két osztály közti egyirányú kapcsolatot jelent
    * Segít az osztályok közti infotmációk megosztását és együttműködését
16. Mi az absztrakt osztály?
    * Metódusai nincsenek implementálva, nem példányosítható
    * Használatához alosztályt kell definiálni
17. Mit csinál a final kulcsszó?
    * Azt jelöli, hogy egy osztálynak nem lehet alosztálya
18. Soroljon fel 4 népszerű objektumorientált programozási nyelvet!
    * Rust
    * C++
    * Python
    * C#
19. Milyen névkonvenciókat kell használni a Java osztály, adattag, metódus és paraméterek definiálásánál?
    * Osztály: A kezdőbetű nagy, neve legyen specifikáció vagy fejléc
    * Adattag: Teljesen kisbetűs, ha több szó akkor egybe írjuk és a 2. szótól kezdve minden szó első betűje nagy
    * Metódus: Teljesen kisbetűs igével kezdődik, ha több szó akkor egybe írjuk és a 2. szótól kezdve minden szó első betűje nagy
20. Mi az a konstruktor? Mi történik, ha egy osztályhoz nem adunk meg konstruktort?
    * Létrehozza az objektumot
    * Visszatérés nélküli, osztály nevét használó metódus
    * Ha nem hozunk létre saját konstruktort, akkor a fordító biztosít egy paraméter nélküli, üres konstruktort
21. Hogyan példányosítunk Javaban egy osztályt?
    * A class nevét kell adattípusként használni
    * Classnev nev = new Classnev();
    * A zárójelben kell megadni a paramétereket, ha hoztunk létre saját konstruktort
    * Etel pizza = new Etel(„Szalámis pizza”, 1, 3000);
    * A példában a konstruktor egy ételnevet, egy azonosítót és egy árat vár el, ez egy string és két integer
22. Java Garbage Collector mit csinál? Mit kell róla tudni?
    * Felszabadítja a memóriából a nem használt objektumokat
    * Automatikusan és időszakosan működik
23. Osztály tagjainak és metódusainak láthatósági módosítói Javaban:
    * public: mindenhonnan elérhető
    * private: csak az örökölt osztályokból és az eredeti osztályból érhető el
    * private: csak az adott osztályból érhető el
24. Javaban a static kulcsszó használata:
    * A static tagok magához az osztályhoz tartoznak, nem pedig csak egy-egy objektumhoz, tehát csak egy van belőle
25. Javaban hogyan deklarálunk konstanst? Névkonvenció is kell.
    * static final adattipus NEV = valami;
    * A névnek teljesen nagybetűsnek kell lennie, ha több szóból áll aláhúzással kell elválasztani őket
    * static final int SZALAMIS\_PIZZA\_ARA = 3500;
26. Mi az a Java csomag? Hogyan adunk neki nevet?
    * A csomag egy névtér, ami egybe organizálja az egymáshoz kapcsolódó osztályokat és interfészeket
    * A kód legelső sorában kell elhelyezni őket a package kulcsszóval
27. A Java csomag elemeinek milyen láthatósági módosítót lehet adni? Melyik mit jelent?
    * public: a csomag látható minden osztálynak mindenhonnan
    * private: alapértelmezett, csak a saját csomagban látható
28. Hogyan lehet használni (meghívni) a Java csomag elemeit?
    * Elemre kell hivatkoznunk (kódon belül) – package.elem
    * Vagy meg kell hívnunk a csomag elemeit – import package.elem
    * Vagy meghívhatjuk az egész csomagot – import package.\*
29. Soroljon fel 4-et a Java API beépített csomagjai közül!
    * java.Math
    * java.lang
    * java.io
    * java.net
30. Mi az annotáció Javaban? Milyen formátuma van?
    * Olyan metadata, amely olyan adatot ad a programnak, ami nem a program része
    * Tudja használni a fordító, hogy hibákat találjon, felfüggessze a figyelmeztetéseket
    * @ jelölő, névtelen is lehet @valami
31. Mit csinál az @Override annotáció Javaban?
    * Szól a fordítónak, hogy a szuper osztályban definiált elem felül lesz írva később egy alosztályban, ezzel megváltoztatva a viselkedését
32. Javaban mit csinal a this kulcsszó? Hol, mire használjuk?
    * Egy konstruktorban vagy metódusban az adott objektumra referál, amire az adott konstruktor/metódus meg lett hívva
    * Az objektum bármely elemére tudunk ezzel hivatkozni
33. Javaban mit csinál a super kulcsszó? Hol, mire használjuk?
    * A szülő osztályra referál
    * Metódusokat, konstruktorokat tudunk meghívni vele, az alosztályban használjuk
34. Miért fontos a Java Object osztálya? Mit kell róla tudni?
    * Minden osztály az Object osztály alosztálya
    * Alapvető viselkedéseket biztosít
    * toString, getObject
    * Nincs szülője
35. Mi a baj a Javaban a String osztállyal? Mit és hogyan használunk helyette?
    * Konkatenálni + -al lehet, konstans szövegeket foglal le, emiatt csak a string buffer változtatható csak
    * Helyette a StringBuildert használjuk, ami gyorsabb teljesítményt biztosít a StringBuffernél
36. Mit kell tudni Javaban a numerikus típusok közötti konverzióról?
    * Kisebb és egyszerűbb konverziók esetén elég átadni az értéket egyik típusról a másikra
    * Egyébként castolást kell alkalmazni
    * float valami = (float) myDouble;
37. Hogyan konvertálunk számot Stringgé és vissza?
    * A wrapper osztályok rendelkeznek toString eljárással, ami az adott értéket visszaadja szövegként
    * Stringet számmá a wrapper osztály parse eljárását kell használni
    * Integer.parseInt(„6”)
38. Paraméterátadás átadás módjáról Javaban mit kell tudni?
    * Érték szerinti paraméterátadás van, kivéve ha objektumot akarunk paraméterként átadni
39. Hogyan lehet tetszőleges számú paramétert áradni egy Javaj metódusnak?
    * Vesszővel elválasztva kell a zárójelbe beírni a paramétereket
    * void valami(int szam1, double szam2, int[] tomb){}
40. Mit csinál Javában a return utasítás?
    * A return utasítás meghívódik, ha a metódus véget ért, meghívjuk vagy exceptiont kapunk
    * Ha eljárást csinálunk, az elején deklaráljuk a visszatérési értéket és kötelező tartalmaznia egy returnt
    * Kilép nekünk az adott blokkból a visszatérési értékkel
41. Inicializáló mező és blokk Javaban. Mire valók, hogyan használjuk őket?
    * Inicializáció mező: a blokkunk legelején pl. deklarálunk egy változót és értéket adunk neki
    * Inicializáció blokk: egy önálló, név nélküli kapcsos zárójel ({}) egy osztályon belül, a lényege, hogy minden objektum létrehozásakor lefut a benne lévő kód, többet is el tudunk helyezni belőle egy osztályon belül
42. Java Enum típus:
    * Speciális adattípus, előre definiált konstansokat tárol
    * Javaban metódusokat és egyéb mezőket is létrehozhatunk benne
43. Java osztályok közötti öröklés:
    * A szülőosztály minden változóját és metódusát megörökli a gyerekosztály, hozzáférni láthatóságtól függően tud
    * Az Object osztálynak nincs szülőosztálya
    * Egyszeres öröklés lehetséges Javaban
44. Java osztályok közötti öröklés esetén a konstruktorok hogyan öröklődnek?
    * A szülőosztály konstruktorai nem öröklődnek, viszont a gyerekosztályok hozzá tudnak férni
45. Java osztályok közötti öröklés esetén mit jelent a metódus felülírása?
    * Egy szülőosztály egy megörökölt metódusának tudjuk módosítani a működését, de a neve, paraméterei és visszatérési értéke változatlan marad
    * @Override annotációval jelezzük a fordítónak
46. Java szuperosztály és alosztály is definiál egy ugyanolyan nevű statikus metódust. Hívható-e, és ha igen, hogyan a szuperosztály metódusa? Ha csak a szuperosztály definiál statikus metódust, akkor az alosztállyal tudjuk-e hívni?
    * Ebben az esetben a gyerekosztály statikus metódusát érjük el, a szülőosztályét elrejti a fordító
47. Java szuperosztály és alosztály is definiál egy ugyanolyan nevű, de más típusú adattagot. Használhatom-e és ha igen, hogyan az alosztályból a szuperosztály adattagját?
    * A szuperosztály metódusait elrejti a fordító, csak a szuperosztály metódusait érhetjük el, kivéve ha a super kulcsszóval referálunk rá
48. Mit jelent a konstruktorok lánca (chain of constructors)?
    * Ha a gyermekosztályban nincsen konstruktor definiálva, akkor a szülő osztály üres konstruktorral hívja meg automatikusan
    * Ha a szülőnek nincs üres konstruktora, akkor a fordítás ideji hibát kapunk
49. Java öröklésnél, metódus felülírásakor a láthatósági módosító változhat-e, és ha igen, hogyan?
    * A gyerekosztályban tudnak változni a szülőosztály láthatóságai, de csak felfele
    * Protectedből lehet public, de private nem
50. Mit jelent Javaban a polimorfizmus?
    * A gyerekosztály rendelkezik saját egyedi tulajdonságokkal, de megörökli a szülőosztály egyes funkcionalitását
51. Java absztakt osztály, asztrakt metódus:
    * Az absztrakt osztályok csak deklarálva vannak, implementálva nem, nem lehet őket példányosítani, de örökölni igen és a gyerekosztályukban vannak implementálva a metódusok
52. Az alkalmazásfejlesztés életciklusának lépései:
    * Vízió
    * Követelmények feltárása
    * Elemzés
    * Architektúrális tervezés
    * Tervezés
    * Implementálás
    * Tesztelés
    * Üzembe helyezés
    * Üzemeltetés
    * Karbantartás
    * Üzemen kívül helyezés
53. Mi az UML?
    * Elemzés és tervezés eszköze, szabványos jelölőrendszer
54. Hogy néz ki az UML osztálydiagram?
    * Négyzetekbe rendezzük az UML-t, legfelül az osztály neve, alatta a változói, alatta a metódusai
    * A statikus tagok és metódusok aláhúzottak
    * private: - protected: # public: +
    * package: ~
55. Az UML hogyan jelöli az osztályok és interfészek közötti öröklést? Mi örökölhet mitől és hány szülő lehet?
    * Az öröklődést nyilakkal jelölik
    * A vonal végén lehet a kapcsolat szerepe, számossága
    * A nyíl jelöli a kapcsolat irányát
    * Kétirányú kapcsolatot nem szokták jelezni
56. Az UML hogyan jelöli az absztrakt osztályt és interfészt?
    * Absztrakt: dőlt betűvel vagy {abstract} megszorítás
    * Interfész: <<interface>>
57. Mi az a Java interfész? Mit tartalmazhat?
    * Osztályok által implementálhatók
    * Interfészek által bővíthetők
    * Alapból absztrakt és public metódusok és punlic static final attribútumok szerepelnek benne
58. Mire szolgál a Java interface default metódusa?
    * Segítik visszafele a kompatibilitást úgy, hogy új metódusokat tudunk hozzáadni az intefészekhez, amik elérhetőek lesznek a már implementált osztályokban
59. Java interfészek közötti öröklés:
    * Egy interfész több interfészt is tud bővíteni
60. Java interfész és osztály közötti öröklés:
    * Egy osztály egyszerre több interfészt is tud implementálni
61. Hogyan lehet használni egy Java interfészt?
    * Absztrakció és többszörös öröklődés érdekében lehet használni egy interfészt
    * Osztály neve után kell írni, hogy implements interfaceneve
62. Java interfészekben definiált default metódusok, ha az öröklés során konfliktusba kerülnek, annak mi lesz a feloldása?
    * Fordítási hibába ütközünk
    * Úgy tudjuk javítani, ha átírjuk a szupertípus metódusát
63. Mi az a Java generikus? Milyen szintaktikával lehet generikust definiálni? Hogyan lehet meghívni generikussal definiált elemet?
    * Lehetővé teszi, hogy paraméterként egy adattípusthasználjunk, amikor interfészeket, osztályokat vagy metódusokat definiálunk
    * Létrehozása: class nev<T1>(){}, T1 bármilyen nem primitív típus lehet
    * Meghívása: new class nev<Integer>()
64. Java generikus metódus, statikus metódus és hívásuk:
    * Saját típussal dolgozunk, amelynek a láthatósága az eljárásra van limitálva
    * Az eljárások lehetnek statikusak, nem statikusak
    * Meghívni az eljaras<T1> -el lehet
65. Java generikus, bounded type:
    * Azt szabjuk meg, hogy a generikus milyen típust fogadhat el
    * Ezt az extends kulcsszóval érhetjük el
    * class nev<T extends B> (B saját osztály)
66. Java generikus, wildcards
    * A generikus programozásban a kérdőjelet (?) wildcardnak hívjuk
    * Ezt tudjuk paraméter, mező vagy változó típusaként használni
67. Java generikus, type erasure
    * Azt biztosítja, hogy a boundjukban legyenek, vagy az Object osztályt kapjuk generikusként
68. Collections osztály
    * Kollekciók ősosztálya
    * Ez tartalmazza a szükséges interfészeket és osztályokat, amik alapból megtalálhatóak a Java API-ban
69. Mi az a kollekció Javaban? A Java API milyen eszközöket biztosít a kollekciókhoz?
    * Egy olyan objektum, ami több elemet csoportosít
    * Arra használjuk, hogy az adatokat tároljuk, elérjük és manipulárjuk
    * A Java interfészek, a kollekciók implementációját és algoritmusokat biztosít
70. A képen szöveg, képernyőkép, Betűtípus, sor látható

    Automatikusan generált leírás Rajzolja le a Java kollekció interfészeinek az öröklési rendszerét!
71. A kollekció elemeinek rendezését hogyan lehet megvalódítani?
    * Collections.sort(myList)
    * Csak abban az esetben, hogyha a kollekciónk rendelkezik a Comparable interfész implementációjával
    * Ellenkező esetlben exceptiont kapunk
72. Comparable interfész:
    * Azt biztosítja, hogy egy objektumot össze tudjuk hasonlítani a saját osztályunkkal, a compareTo eljárással
73. Comparator interfész:
    * Azt biztosítja, hogy egy osztály két elemét össze tudjuk hasonlítani
74. Java beáagyazott osztály:
    * Osztályon belül tudunk létrehozni egy beágyazott osztályt
    * Ha nem statikus, akkor elérhetjük a külső osztály változóit és metódusait, akkor is, ha privátok
    * Tudjuk láthatósággal is illetni
75. Statikus beágyazott osztály:
    * Nem férnek hozzá a külső osztály elemeihez
    * Csak objektumon keresztül tudjuk elérni
76. Java local class:
    * Egy osztályon belüli blokkok között lett létrehozva
    * Tartozhat eljáráshoz, inicializációs blokkhoz, loophoz, vagy if elágazáshoz
    * El tudja érni a bezáró osztályának elemeit
    * Nem lehet láthatóságot adni, de final és abstract lehet
77. Java anonymous class:
    * Egy olyan belső class, aminek nem adunk nevet és csak egy object készül belőle
    * Csak egy interfészt tud egyszerre implementálni
    * Nem tudunk konstruktort írni hozzá
78. Java funkcionális interfész:
    * Csak egyetlen metódust tartalmaznak
    * @FunctionalInterface annotációval tudjuk jelezni
79. Soroljon fel 5 beépített Java funkcionális interfészt!
    * Runnable – run()
    * Comparable – compareTo()
    * ActionListener – actionPerformed()
    * Callable – call()
    * Supllier – get()
80. Java lambda kifejezés:
    * Rövid blokkokból állnak, paramétereket fogadnak el és értéket adnak vissza
    * Hasonlítanak az eljárásokhoz, de nem adunk nekik nevet és a metódus blokkjában azonnal lehet implementálni őket
81. Java metódus referencia:
    * Arra tudjuk használni, hogy a kódot újra tudjuk használni
    * Object::eljarasnev
82. Javadoc mit csinál, mire való, mire használjuk?
    * Kódunk saját dokumentációja HTML formátumban
    * /\*\* \*/ tagek között adjuk meg
    * Bármilyen osztály, metódus vagy mező felett el tudjuk helyezni
    * @-al metadata-t tudunk specifikálni
83. Sololjon fel 5 Javadoc taget és mutassa be, hogy mit csinálnak!
    * @author, @version – csak osztályoknál és interfészeknél használatos, a szerzőt és a verziót írja le
    * @param – csak metódusoknál és konstruktoroknál használatos, paramétereket írja le
    * @return – csak metódusoknál használatos, a visszatérési értéket írja le
    * @exception – lehetséges hibákat írja le
84. Mi az a kivétel?
    * Egy olyan esemény, ami megszakítja a kód futását, mert hibába ütközött
    * Amikor hibába ütközünk az éppen aktuális metódusunk létrehoz egy exception objectet, ami a hibáról tartalmaz információkat
85. A kivételeknek mi a 3 alapvető kategória? Melyikről mit kell tudni?
    * Checked Exception: egy jól megírt programnak nem kellene gondot okoznia, ellenőrzött hibáknak hívjuk őket
    * Error: külső hiba, amivel a program nem tud mit kezdeni
    * Runtime Exception: belső hiba, amivel a program nem tud mit kezdeni
86. Hogy néz ki a try-catch-final utasítás és melyik része mit csinál?
    * try {mit próbáljon meg}
    * catch (ExceptionType nev) {mi történjen a hibánál}
    * a final blokk az egész végén fut le, ha a try blokknak vége, akár beléptünk a catch blokkba akár nem
87. Mit csinál a try-with-resource utasítás?
    * Azt biztosítja, hogy a fájlműveletek biztonságosan befejeződtek
    * Automatikusan bezárja őket
88. Hogyan lehet kivételt dobni?
    * throw utasítással
    * Egyetlen objektumot vár el
89. Hogyan hozhatunk létre Java kivétel osztályt?
    * Ha saját kivételt akarunk létrehozni, akkor az Error vagy az Exception osztályból kell örököltetnünk
90. Mi az az I/O Stream?
    * Reprezentálja a bejövő forrást és a kimenő célt
    * Lehetővé teszi az adatok folyamatos írását és olvasását
91. Mi az a Byte Stream?
    * 8 bites I/O művelet
92. Mi az a Character Stream?
    * Automatikusan átkonvertálja a megfelelő kódolásra a szöveget
    * Általában 8 bites ASCII
93. Scanning and Fromatting
    * Hogy az ember számára elérhető adattal tudjunk dolgozni, a Java biztosít erre egy API-t, ami ezt elősegíti
    * A Scanner API a beviteli adatok beolvasását teszi lehetővé, például billentyűzetről vagy fájlból, majd feldarabolja az inputot
    * A Formatting Api olvashatóvá teszi azt számunkra a megjelenítés formázásával
94. Mi a 3 standard stream a Javaban? Melyik mire való?
    * Standard input stream: szöveget olvas stdiről, azaz a program bemenetét olvassa be, például billentyűzetről vagy fájlból
    * Standard output stream: szöveget ír ki stdora, azaz a program kimenetét írja ki, például képernyőre vagy fájlba
    * Standard error stream: a program hibájáit és hibaüzeneteit írjuk le vele
95. Mi a data stream?
    * Bináris I/O műveleteket biztosít a primitív értékekhez
    * Alapvető adattípusokat és objektumokat tud kezelni
    * Segítségével az adatokat bináris formában lehet írni és olvasni
96. Mi az object stream?
    * Az összetett típusainkank biztosítja az I/O műveleteket
97. Mi az a buffered stream?
    * Segít javítani az adatok írásának és olvasásának hatékonyságát
    * Az adatok egy bufferbe olvassák be vagy írják ki
    * A bufferből végzik az olvasásokat és írásokat
98. Javadoc eszköz használata esetén a fő leírást hova írja? Milyen elemekhez adhat meg dokumentációt?
    * A fő leírást a metódusok, modulok, csomagok, stb. elé adjuk meg
    * Csak egy dokumentációnk lehet
99. Mire való a Junit?
    * Egy olyan unit tesztelési framework Javahoz, ami annyit jelent, hogy apróbb részletekben teszteljük a kódot egy várt bemenettel és kimenettel
100. Solroljon fel a Junit eszközben használatos annotációkból 5-öt és magyarázza el mire valók!
     * @Test – az alapértelmezett teszt esetet kicseréli az általunk megadott teszt esetre
     * @Before – akkor adjuk meg, ha a tesztünk előtt akarunk valamit lefuttatni
     * @BeforeClass – az összes teszt eset előtt fut le ez a rséz
     * @After – a tesztünk után lefuttatandó részhez adjuk meg
     * @AfterClass – az összes teszt eset lefuttatása után fut le ez a rész
101. A Junit eszköz assertionjai mire valók? Soroljon fel belőlük 10-et és magyarázza őket!
     * A teszthez nagyon hasznos metódusok, azt ellenőrzik, hogy helyes választ kapunk-e
     * Csak akkor rögzíti a válaszokat, ha hibás választ kapunk
     * assertEqueals(bool exc, bool act) – két értéket hasonlít össze
     * assertTrue(bool c) – ellenőrzi, hogy igaz-e a feltételünk
     * assertFalse(bool c) – ellenőrzi, hogy hamis-e a feltételünk
     * assertNull(object o) – ellenőrzi, hogy a feltételünk NULL-e
     * assertNotNull(object o) – ellenőrzi, hogy a feltételünk nem NULL-e
     * assertSame(object1, object 2) – ellenőrzi, hogy a feltételeink azonosak-e
     * assertNotSame(object1, object 2) – ellenőrzi, hogy a feltételeink nem azonosak-e
     * assertArrayEqueals(arr1, arr2) – ellenőrzi, hogy a két tömb azonos-e
     * assertThat(T t, M m) – ellenőrzi, hogy a t eleget tesz-e az m-nek
     * fail() – elbuk egy tesztet üzenet nélkül
102. A kollekciók aggregáló műveletei esetén a pipelinenak milyen részei vannak? Melyik mire való?
     * source – lehet kollekció, array vagy I/O
     * intermediate operators – ezek a filterek, amelyek új stringet generálnak
     * terminal operators – nem stream beli eredményt kapunk vissza
103. Hogyan működik a reduce művelet a kollekciók aggregáló műveletei esetén?
     * Mindig egy új értéket ad vissza, értékek megadása után egy lambda funkcióval tudjuk megadni, hogy mi legyen a szabályunk
104. Hogyan működik a collect a kollekciók aggregáló műveletei esetén?
     * Módosít egy már létező értéket
105. Mutassa be a kollekciók aggregáló műveleteiből a groupingBy és a reducing műveleteket!
     * groupingBy – egy mapet ad vissza, ami egy lambda érték alapján szűri az értékeket
     * reducing – egyetlen eredményt generál több értékben 3 feltétel alapján
106. A JAR eszköz mire való, milyen műveleteket lehet vele elvégezni?
     * Több fájlt egybe csomagol, hogy egyetlen futtatható programunk lehessen
107. A JAR fájl mainfestjét mutassa be!
     * A manifest fájl metaadatokat tartalmaz
     * Ezt érjük el először, e nélkül nem fut le a program
     * Csak egy lehet belőle
108. Mi a Java modul?
     * Külön csomagolt java packagek, amik modulárisan beépíthetők
     * Igényelhetnek további modulokat a futtatáshoz
109. A Java modul module-info.java állományában mit jelentenek az exports, exports … to, requires, uses direktívák?
     * exports – a modul csomagjainak a public funkciót és paramétereit írja le, hogy hogyan érhetők el
     * exports … to – vesszőkkel elválasztva tudjuk megadni azt, hogy mely csomagokat exportáljuk
     * requires – definiálja, hogy milyen másik moduloktól függ a modulunk működése
     * uses – a szükséges absztrakt osztályokat sorolja fel
110. Mi az a Project Lombok?
     * Egy Java könyvtár, ami egy IDE kiegészítőként működik, rengeteg dolgot automatizál
111. Project Lombokkal hogyan ad meg gettert, settert?
     * Annotációval - @Getter, @Setter
112. Project Lombokkal hogyan ad meg toStringet?
     * A lombok.ToString importálása után a @ToString annotációval a class előtt
113. Project Lombokkal hogyan ad meg equalst és HashCodeot?
     * lombok.EqualsAndHashCode importálása után @EqualsAndHashCode annotációval