01001. 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 eszközöket
- A Java Runtime Environment segíti betölteni ezeket a JVM számára, hogy futtatni tudja

01002. JVM: mi az, miért fontos?

- Java Virtual Machine
- Ezen futnak a java kódok
- A java platform része
- Ettől platform független a nyelv

01003. Oldja fel és magyarázza, mire lehet használni: JRE, JDK, IDE

- JRE: Java Runtime Environment - kapcsolatot biztosítja a java és az operációs rendszer között; memóriához való hozzáférés, IO műveletek
- JDK: Java Development Kit – a szükséges eszközöket biztosítja a java fejlesztéshez
- IDE: Integrated Development Environment – szövegszerkesztő, ami rengeteg hasznos funkcióval rendelkezik a kódunk megírásához, a szoftverfejlesztés folyamatát gyorsítja pl. Eclipse, IntelliJ

01004. 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 parancssori argumentumok listája

01005. Sorolja fel és jellemezze a primitív adattípusokat és a wrapper osztályaikat!

- Adatokat tudunk velük tárolni, különböző típusai vannak
- Egész számot 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

01006. String osztály

- A String egy karakter tömb
- ’’ ” között lehet megadni
- Objectnek számít
- Deklaráláskor új objectet készít pl. String nev = ’’hello”;

01007. Tömb (array) deklarációja, használata

- A tömb N darab érték lefoglalása egymás mellett
- Deklaráláskor az adattípus mögé []-t írunk, utána new adattípus[N]-ként tudjuk lefoglalni a memóriában pl. int[] valtozo = new int[6]
- valtozo[N]-ként tudunk referálni arra, hogy hanyadik elemet szeretnénk

02009. 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

02010. Absztrakció fogalma

- A modellezésnél fontos tulajdonságok, viselkedések leírása

02011. Mit jelent az egységbezárás

- Szabályozza, hogy egy object egyes elemeit miként érhetjük el
- El tudunk vele rejteni adattagokat, függvényeket, eljárásokat

02012. Mi a különbség az objektum állapota és viselkedése között?

- Állapota: attribútumait értjük
- Viselkedése: metódusait, függvényeit értjük

02013. Mi az osztályattribútum és osztálymetódus?

- Az objektum állapota és viselkedése helyett osztályszinten tudjuk vizsgálni az attribútumokat és metódusokat

02014. 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

02015. Mi az öröklés (általánosan)?

- Az öröklődés olyan kapcsolattípus, ahol lehetővé tesszük az osztályok tulajdonságainak és metódusainak öröklődését

02016. Mi az aggregáció?

- Egy osztály egy másik osztály attribútumait tartalmazza
- Kapcsolatuk laza, nem feltétlen kölcsönös

02017. Mi a kompozíció?

- Egy osztály egy másik osztály attribútumait tartalmazza
- Kapcsolatok szoros

02018. Mi az asszociáció?

- Két osztály közti kétirányú kapcsolatot jelenti
- Segíti az osztályok közti információk megosztását és együttműködését

02019. Mi az absztrakt osztály?

- Az absztrakt osztály metódusai nincsenek implementálva, nem példányosítható
- Használatához alosztályt kell definiálni

02020. Mit csinál a final kulcsszó?

- Azt jelöli, hogy egy osztálynak nem lehet alosztálya

02021. Soroljon fel 4 népszerű objektumorientált programozási nyelvet!

- Rust
- C++
- Python
- C#

03023. 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 kezdődően minden szó első betűje nagy
- Metódus: teljesen kisbetűs igével kezdődik, ha több szó akkor egybe írjuk és a 2. szótól kezdődően minden szó első betűje nagy

03024. Mi az a konstruktor? Mi történik, ha egy osztályhoz nem adunk meg konstruktort Javaban?

- Létrehozza az objektumot
- Visszatérés nélküli, osztály nevét használó metódus
- Ha nem hozunk létre saját konstruktort, akkor a fordító biztosít egy paraméter nélküli üres konstruktort

03025. Hogyan példányosítunk Javaban egy osztályt?

- A class nevét kell adattípusként használni pl. Classnev nev = new Classnev();

03026. Java Garbage Collector mit csinál? Mit kell róla tudni?

- Felszabadítja a memóriából a nem használt objektumokat
- Automatikusan és időszakosan működik

03027. Osztály tagjainak és metódusainak láthatósági módosítói Javaban

- public: mindenhonnan elérhető
- protected: csak örökölt alosztályból és eredeti osztályból érhetők el
- private: csak az adott class-ból érhető el

03028. Javaban a static kulcsszó használata

- A static tagok magához az osztályhoz tartoznak, nem pedig csak egy-egy objektumhoz, tehát csak egy van belőle

03029. Javaban hogyan deklarálunk konstanst? Névkonvenció is kell.

- static final adattipus NEV = valami;
- a névnak teljesen nagybetűsnek kell lennie, ha több szóból áll, akkor aláhúzással kell elválasztani őket

04030. 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

04031. 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

04032. 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 meghívni az egész csomagot – import package.*

04033. Soroljon fel 4-et a Java API beépített csomagjai közül!

- java.Math
- java.lang
- java.io
- java.net

04034. (Java) Mi az az annotáció? 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 figyelmesztetéseket
- @ jelölő, névtelen is lehet pl. @valami

04035. Mit csinál az @Override annotáció Javaban?

- Szól a fordítónak, hogy a super osztályban definiált elem felül lesz írva később egy alosztályban, ezzel megváltoztatva a viselkedését

05036. Javaban mit csinál a this kulcsszó? Hol használjuk, mire?

- Egy konstruktorban vagy metódusban az adott objektumra referál, amire az adott konstruktor/metódus meg lett hívva
- Az objektum bármely elemére tudunk ezzel hivatkozni

05037. Javaban mit csinál a super kulcsszó? Hol használjuk, mire?

- A szülő osztályra referál
- Metódusokat, konstruktorokat tudunk meghívni vele

05038. Miért fontos a Java Object osztálya? Mit kell róla tudni?

- Minden osztály az Object osztály alosztálya
- Alapvető viselkedéseket biztosít pl. toString, getObject
- Nincsen szülője

05039. (Java) Mi a baj a String osztállyal? Mit és hogyan használunk helyette?

- Konkatanálni a +-al lehet
- Konstans szövegeket foglal le
- A string buffer változtatható csak
- A StringBuilder a string buffert helyettesíti
- Gyorsabb teljesítményt biztosít

05040. (Java) Mit kell tudni a numerikus típusok közötti konverzióról?

- Kisebb és egyszerűbb konverziók esetén elég átadni az értéket egyik típusról a másikra
- Egyébként castolást kell alkalmazni pl. float valami = (float)myDouble;

05041. (Java) 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 pl. Int.parseInt(''6'')

05042. (Java) paraméterátadás módjáról mit kell tudni?

- Érték szerinti paraméterátadás van, kivéve ha objektumot akarunk paraméterként átadni

05043. Hogyan lehet tetszőleges számú paramétert átadni egy Java metódusnak?

- Vesszővel elválasztva kell a zárójelbe beírni a paramétereket pl. void valami(int szam1, double szam2, int[] tomb){}

05044. Mit csinál Javaban a return utasítás?

- A return utasítás meghívódik, ha a metódus véget ért, meghívjuk vagy exception-t kapunk
- Ha eljárást csinálunk, az elején deklaráljuk a visszatérési értéket és kötelező tartalmaznia egy return-t
- Kilép nekünk az adott blokkból a visszatérési értékkel

05045. Inicializáló mező és blokk Javaban. Mire valók, hogyan használjuk őket?

- Inicializáló mező: a blokkunk legelején pl. deklarálunk egy változót és értéket adunk neki
- Inicializáló blokk: egy önálló, név nélküli {} egy osztályon belül; a lényege, hogy minden objektum létrehozásakor lefut a benne lévő kód, többet is el tudunk helyezni belőle egy osztályon belül

05046. Java Enum típus

- Speciális adattípus, előre definiált konstansokat tárol
- Java-ban metódusokat és egyéb mezőket is létrehozhatunk benne

06047. Java osztályok közötti öröklés

- A szülőosztály minden változóját és metódusát megörökli a gyerekosztály, hozzáférni a láthatóságtól függően tud
- Az Object osztálynak nincs szülőosztálya
- Egyszeres öröklődés lehetséges Java-ban

06048. Java osztályok közötti öröklés esetén a konstruktorok hogyan öröklődnek?

- A szülőosztály konstruktorai nem öröklődnek, viszont a gyerekosztályok hozzá tudnak férni

06049. Java osztályok közötti öröklés esetén mit jelent a metódus felülírása?

- Egy szülőosztály egy megörökölt metódusának tudjuk módosítani működését, de a neve, paramétere és visszatérési értéke változatlan marad
- @Override annotációval jelezzük a fordítónak

06050. 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

06051. 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 gyerekosztály metódusait érhetjük el, kivéve ha a super kulcsszóval referálunk rá

06052. Mit jelent a konstruktorok lánca (chain of constructor)

- Ha a gyermekosztályban nincsen konstruktor definiálva, akkor a szülő osztály üres konstruktorral hívja meg automatikusan
- Ha a szülőnek nincs üres konstruktora, akkor fordítás ideji hibát kapunk

0653. Java öröklésnél, metódus felülíráskor a láthatósági módosító változhat-e, és ha igen, hogyan?

- A gyerekosztályban tudnak változni a szülőosztály láthatóságai de csak felfele pl. protectedből lehet public, de private nem

06054. Mit jelent Javaban a polymorfizmus?

- A gyerekosztály rendelkezik saját egyedi tulajdonságokkal, de megörökli a szülőosztály egyes funkcionalitását

06055. Java absztrakt osztály, absztrakt metódus

- Az absztrakt osztályok csak deklarálva vannak, implementálva nem, nem lehet őket példányosítani, de örökölni igen és a gyerekosztályukban vannak implementálva a metódusok

07056. Az alkalmazásfejlesztés életciklusának lépései (felsorolás elég)

- Vízió
- Követelmények feltárása
- Elemzés
- Architektúrális tervezés
- Tervezés
- Implementálás
- Tesztelés
- Üzembe helyezés
- Üzemeltetés
- Karbantartás
- Üzemen kívül helyezés

07057. Mi az az UML?

- Elemzés és tervezés eszköze, szabványos jelölőrendszer

07058. Hogy néz ki az UML osztálydiagram?

- Négyzetekbe rendezzük az UML-t, lefelül az osztály neve, alatta a változói, alatta az eljárásai
- A statikus tagok és metódusok aláhúzottak
- private: - , protected: #, public: +
- package: ~

07059. 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

07060. Az UML hogyan jelöli az absztrakt osztályt és az interfészt?

- Absztrakt: dőlt betűvel vagy { abstract } megszorítás
- Interfész: <<interface>>

08061. Mi az a Java interfész? Mit tartalmazhat (csak felsorolás)?

- Osztályok által implementálhatók
- Interfészek által bővíthetők
- Alapból absztrakt és public metódusok és public static final attribútumok szerepelnek benne

08062. Mire szolgál a Java interfész default metódusa?

- Segítik visszafele a kompatibilitást úgy, hogy új metódusokat tudunk hozzáadni az interfészekhez, amik elérhetőek lesznek a már implementált osztályokban

08063. Java interfészek közötti öröklés

- Egy interfész több interfészt is tud bővíteni

08064. Java interfész és osztály közötti öröklés

- Egy osztály egyszerre több interfészt is tud implementálni

08065. Hogyan lehet használni egy Java interfészt?

- Absztrakció és többszörös öröklődés érdekében lehet használni egy interfészt
- Osztály neve után kell írni, hogy implements interfaceneve

08066. Java interfészekben definiált default metódusok ha az öröklés során konfliktusba kerülenk, annak mi lesz a feloldása?

- Fordítási hibába ütközünk
- Úgy tudjuk javítani, ha átírjuk a szupertípus metódusát

09067. Mi az a Java generikus? Milyen szintaktikával lehet generikust definiálni? Hogyan lehet meghívni generikussal definiált elemet?

- Lehetővé teszi, hogy paraméterként egy adattípust használjunk, amikor interfészeket, osztályokat vagy metódusokat definiálunk
- Létrehozása: pl. class nev<T1>(){}, T1 bármilyen nem primitív típus lehet
- Meghívása: pl. new class nev<Integer>()

09068. 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

09069. Java generikus, bounded type

- Azt szabjuk meg, hogy a generikus milyen típust fogadhat el
- Ezt azt extends kulcsszóval érhetjük el pl. class nev<T extends B> (B saját osztály)

09070. Java generikus, wildcards

- A generikus programozásban a ?-t wildcard-nak hívjuk
- Ezt tudjuk paraméter, mező vagy változó típusaként használni

09071. Java generikus, type erasure

- Azt biztosítja, hogy a bound-jukban legyenek, vagy az object osztályt kapjuk generikusunkként

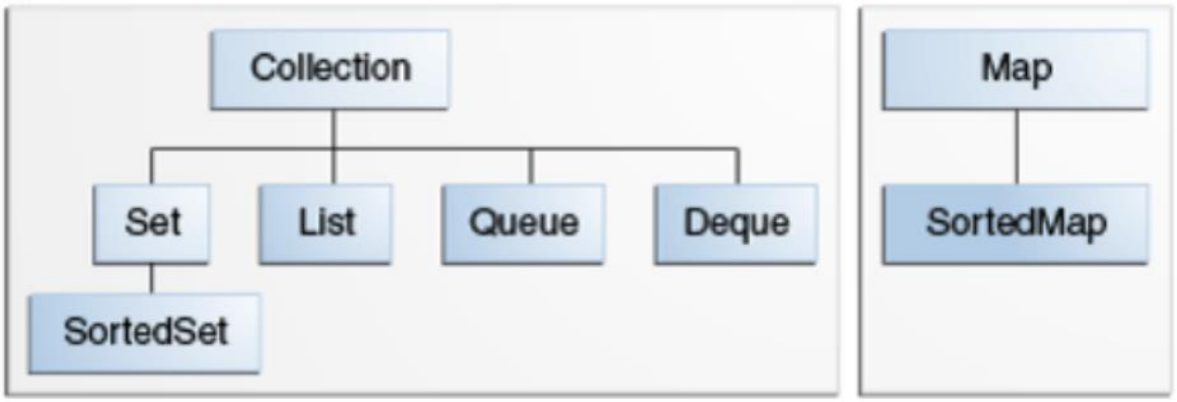
09077. Collections osztály

- Kollekciók ősosztálya
- Ez tartalmazza a szükséges interfészeket és osztályokat, amik alapból megtalálhatóak a Java API-ban

09072. Mi a kollekció (Javaban)? A Java API milyen eszközöket biztosít a kollekciókhoz?

- Egy olyan object, ami több elemet csoportosít
- Arra használjuk, hogy az adatokat tároljuk, elérjük és manipuláljuk
- A java interfészek, a kollekciók implementációját és algoritmusokat biztosít

09073. Rajzolja le a Java kollekció interfészeinek az öröklési rendszerét (a mappel együtt)!



09074. A kollekció elemeinek a rendezését hogyan lehet megvalósítani?

- Collections.sort(myList)
- Csak abban az esetben, hogyha a kollekciónk rendelkezik a Comparable interfész implementációjával
- Ellenkező esetben exception-t kapunk

09075. Comparable interfész

- Azt biztosítja, hogy egy objectet össze tudjunk hasonlítani a saját osztályunkkal, a compareTo eljárással

09076. Comparator interfész

- Azt biztosítja, hogy egy osztály két elemét össze tudjuk hasonlítani

10078. Java beágyazott 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

10079. Statikus beágyazott osztály

- Nem férnek hozzá a külső osztály elemeihez
- Csak objektumon keresztül tudjuk elérni

10080. Java local class

- Egy osztályon belüli blokkok között lett létrehozva
- Tartozhat eljáráshoz, inicializálós blokkhoz, loop-hoz vagy if elágazáshoz
- El tudja érni a bezáró osztályának elemeit
- Nem lehet láthatóságot adni, de final és abstract lehet

10081. 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

11082. Java Funkcionális interfész

- Csak egyetlen metódust tartalmaznak
- @FunctionalInferface-el tudjuk jelezni

11083. Soroljon fel 5 beépített Java funkcionális interfészt

- Runnable – run()
- Comperable – compareTo()
- ActionListener – actionPerformed()
- Callable – call()
- Supplier – get()

11084. 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

11085. Java metódus referencia

- Arra tudjuk használni, hogy a kódot újra tudjuk használni
- Object::eljarasnev

12086. Javadoc mit csinál, mire való, miért használjuk?

- Kódunk saját dokumentációja
- /** */ között adunk meg
- Bármilyen osztály, metódus vagy mező felett el tudjuk helyezni
- @-al metadata-t tudunk specifikálni

12087. Soroljon fel 5 Javadoc tag-et, és mutassa be, hogy mit csinálnak.

- @author, @version – csak classoknál és interfészeknél használatos, szerzőt és a verziót írja le
- @param – csak metódusoknál és konstruktoroknál használatos, paramétereket írja le
- @return - csak metódusoknál használatos, visszatérési értéket írja le
- @exception – lehetséges hibákat írja le

13088. 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

13090. A kivételeknek mi a 3 alapvető kategóriája? Melyikről mit kell tudni?

- Checked exception: egy jól megírt programnak nem kellene gondot okoznia, ellenőrzött hibának hívjuk őket
- Error: külső hiba, amivel a program nem tud mit kezdeni
- Runtime exception: belső hiba, amivel a program nem tud mit kezdeni

13091. Hogy néz ki a try-catch-final utasítás, és melyik része mit csinál?

- try{ hogy mit próbáljon meg }
- catch (ExceptionType nev){ mi történjen az adott hibánál }
- a final blokk az egész végén fut le, ha a try blokknak vége

13092. Mit csinál a try-with-resource utasítás?

- Azt biztosítja, hogy a fájlműveletek biztonságosan befejeződtek
- Automatikusan bezárja őket

13093. Hogyan lehet kivételt dobni?

- throw utasítással
- Egyetlen objektet vár el

13094. Hogyan hozhatunk létre Java kivétel osztályokat?

- Ha saját kivételt akarunk létrehozni, akkor az error vagy az exception osztályokból kell örököltetnünk

14095. Mi az az I/O Stream?

- Reprezentálja a bejövő forrást és a kimenő célt

14096. Mi az a byte stream?

- 8 bites I/O művelet

14096. Mi az a character stream?

- Automatikusan átkonvertálja a megfelelő kódolásra a szöveget
- Általában a 8 bites ASCII

14097. Scanning and Formatting

- Hogy az ember számára érthető adattal tudjunk dolgozni, a java biztosít erre egy API-t, ami ezt elősegíti
- A scanner API feldarabolja az inputot
- A formatting API olvashatóvá teszi azt számunkra

14098. Mi a 3 standard stream a Java-ban? Melyik mire való?

- Standart input stream: szöveget olvas stdi-ről
- Standart output stream: szöveget ír ki stdo-ra
- Standart error stream: a program hibáját írjuk le vele

14099. Mi a data stream?

- Bináris I/O műveleteket biztosít a primitív értékekhez

14100. Mi az object stream?

- Az összetett típusainknak biztosítja az I/O műveleteket

11. Mi az az I/O stream?

- reprezentálja a bejövő forrást és a kimenő célt
- lehetővé teszi az adatok folyamatos írását és olvasását

12. Mi az a byte stream?

- 8 bites I/O művelet

13. Mi az a character stream?

- átkonvertálja megfelelő kódolásra a szöveget

14. 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

15. Scanning and formatting

- Scanning: a beviteli adatok beolvasását teszi lehetővé pl. billentyűzetről vagy fájlból
- Formatting: kimeneti adatok megjelenítésének stílusa és formája

16. Mi a 3 standard stream Java-ban? Melyik mire való?

- standard input stream – a program bemenetét olvassa be, pl. billentyűzetből vagy fájlból
- standard output stream – a program kimenetét írja ki pl. képernyőre vagy fájlba
- standard error stream – a program hibáit és hibaüzeneteit írja ki

17. Mi a data stream?

- alapvető adattípusokat és objektumokat tud kezelni
- segítségével az adatokat bináris formában lehet írni és olvasni

18. Mi az object stream?

- az összetett típusainknak biztosítja az I/O műveleteket

21. Mit csinál a Javadoc eszköz?

- a Javadoc a kódunk saját dokumentációja HTML formátumban
- /** */ tagek között adunk meg
- bármilyen class, metódus vagy mező felett el lehet helyezni

22. Soroljon fel 4 Javadoc tag-et a Javadoc eszközhöz, és magyarázza, hogy mit csinálnak!

- @param – paraméterek dokumentálására szolgál, tartalmazza a nevét és egy leírást, hogy mire szolgál
- @return – a visszatérési érték dokumentálására szolgál, tartalmazza a típusát és egy leírást, hogy mit jelent az érték
- @see – hivatkozást tesz lehetővé más osztályokra, metódusokra
- @exceptions – kivételek dokumentálására szolgál, tartalmazza a paraméter típusát és leírást az okokról vagy a kivétel kezeléséről

23. 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

31. Mire való a JUnit?

- egy olyan unit tesztelési framework a java-hoz, ami annyit jelent, hogy apróbb részletekben teszteljük a kódot egy várt bemenettel és kimenettel

32. Soroljon fel a JUnit eszközben használatos annotációkból 5-öt és magyarázza el mire valók!

- @Test – az alapértelmezett teszt esetet kicseréli az általunk adott teszt esetre
- @Before – akkor adjuk meg, ha a tesztünk előtt akarunk valamit lefuttatni
- @BeforeClass – az összes teszt eset előtt fut le ez a rész
- @After – a tesztünk után lefuttatandó részhez adjuk meg
- @AfterClass – az összes teszt eset lefutása után fut le ez a rész

33. A JUnit eszköz asseration-jai 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
- 1. assertEquals(bool exc, bool act) – két értéket hasonlít össze
- 2. assertTrue(bool c) – ellenőrzi, hogy igaz e a feltételünk
- 3. assertFalse(bool c) – ellenőrzi, hogy hamis e a feltételünk
- 4. assertNull(object o) – ellenőrzi, hogy a feltételünk NULL e
- 5. assertNotNull(object o) – ellenőrzi, hogy a feltételünk nem NULL e
- 6. assertSame(object1, object2) – ellenőrzi, hogy a feltételeink azonosak e
- 7. assertNotSame(object1, object2) – ellenőrzi, hogy a feltételeink nem azonosak e
- 8. assertArrayEquals(arr1, arr2) – ellenőrzi, hogy a két tömb azonos e
- 9. assertThat(T t, M m) – ellenőrzi, hogy a t eleget tesz e az m-nek
- 10. fail() – elbuk egy tesztet üzenet nélkül

41. A kollekciók aggregáló műveletei esetén a pipeline-nak milyen részei vannak? Melyik mire való?

- source – lehet kollekció, array vagy I/O
- intermediate operators – ezek filterek, amelyek új stringet generálnak
- terminal operators – nem stream beli eredményt kapunk vissza

42. 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

43. Hogyan működik a collect a kollekciók aggregáló műveletei esetén?

- módosít egy már létező értéket

44. Mutassa be a kollekciók aggregáló műveleteiből a groupingBy és a reducing műveleteket!

- groupingBy – egy map-et ad vissza, ami egy lambda érték alapján szűri az értékeket
- reducing – egyetlen eredményt generál több értékben 3 feltétel alapján

51. A JAR eszköz mire való, milyen műveleteket lehet vele elvégezni?

- több file-t egybe csomagol, hogy egyetlen futtatható programunk lehessen

52. A JAR file ’’manifest’’-jét mutassa be!

- a manifest file metaadatokat tartalmaz
- ezt érjük el először, e nélkül nem fut le a program
- csak egy lehet belőle

61. Mi a Java modul?

- külön csomagolt java package-ek, amik modulárisan beépíthetők
- igényelhetnek további modulokat a futtatáshoz

62. A Java modul module-info.java állományában mit jelentenek az exports, exports … to, requires, uses direktívák?

- exports – a modul csomagjainak a public funkcióit 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

71. Mi az a Project Lombok?

- egy Java könyvtár, ami a egy IDE kiegészítőként működik, rengetek dolgot automatizál

72. Project Lombok-kal hogyan ad meg getter-t, setter-t?

- annotációval - @Getter, @Setter

73. Project Lombok-kal hogyan ad meg toString-et?

- a lombok.ToString importálása után a @toString annotációval a class előtt

74. Project Lombok-kal hogyan ad meg equals-t és HashCode-ot?

- lombok.EqualsAndHashCode importálása után @EqualsAndHashCode annotációval