01. Mi a legabsztraktabb típus a JAVA-ban?

Interface, tiszta “absztrakt osztály”

Csak a metódusok prototípusa, nincs implementáció

– Név

– Paraméter szignatúra

– Visszatérési érték

• Minden metódus implicite

– public

– abstract

(http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/users/smidl/oktatas/oop/java8.pdf)

// Nem értem miért miskolci anyagot linkel be valaki...

// http://pnyf.inf.elte.hu/kto/teaching/java/material/04_oop.pps

02. Mikor preferált interface-ek használata a konkrét típusok helyett?

Interface-t akkor érdemes használni, ha mi csak egy közvetítő réteget szeretnénk használni, azaz a tényleges megvalósítása az interfaceben felsorolt függvényeknek minket nem érdekel.

// Amennyiben több olyan származtatott oszály lesz, amely bizonyos metódusaiban azonos**???**

03. Mi a különbség az extends és az implements között. Mikor melyiket kell használni?

extends: Egy gyerekosztály ősosztályból való származtatása esetén használandó.

class A extends B { …} esetén az A gyerekosztály a B szülőosztály minden **nem private** függvényét és adattagját látja, módosíthatja (azaz a függvényeket felülírhatja - @Override)

implements: Egy B interface megvalósításakor az implements keyword-öt kell használni. Ez csak interface implementálásakor használható. Míg az extends esetén a keyword jobb oldalára egyetlen osztályt írhatunk csak, az implements keyword után vesszővel elválasztva több interface-t is felsorolhatunk.

class A implements B, C, D { … }

04. Van e többszörös öröklődés JAVA-ban?

NINCS [Maugli: Csak single inheritance van] (többszörös öröklődés: amikor több osztályból örököl az osztályod, de a javaban csak egyet tudsz pl extendelni)

Kiegészítés (csataci): extends (azaz tényleges öröklés) csak egyetlen osztályból lehet, azonban interface-t többet is megvalósíthat (implements) - forrás: http://csis.pace.edu/~bergin/patterns/multipleinheritance.html

05. Sorold fel a primitív típusokat!

byte : [-128..127] (8 bit : 1 bájt)

short : [-32768..32767] (16 bit : 2 bájt)
int : [-2,147,483,648..2,147,483,647] (32 bit : 4 bájt)
long : [-9,223,372,036,854,775,808..9,223,372,036,854,775,807] (64 bit : 8 bájt)
float :
double
boolean : Logikai érték: Igaz/hamis
char :

(http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html)

06. Mire használjuk a Comparable interface-t?
Ha egy osztály implementálja a Comparable interfészt, akkor lehetőségünk van a compareTo függvény overrideolására, ami megadja az adott objektum másik objektumhoz való viszonyát (összehasonlítását). Létezik generikus változata is (*Comparable<T>*), ekkor a compareTo nem Object típusú objektumot vár, hanem az adott T típusút.

07. Mi a különbség static és nem static metódusok között?
A nem static függvények egy adott objektumhoz (példányhoz) tartoznak, és azokat csak objektumokon keresztül tudjuk meghívni. Ezek látják, módosíthatják az adott példányhoz tartozó adattagokat.
A static függvények hívhatóak objektumokon keresztül, de hívhatóak példány nélkül is, közvetlenül az osztályon keresztül, tehát ezek a függvények nem objektumhoz tartoznak, azaz az osztályon belüli nem-static adattagokat nem látják (mivel azok egy-egy példányhoz tartoznak).
08. Mi az összefüggés az objektum statikus és dinamikus típusa között?
Statikus/dinamikus típus: statikus, amivel definiálva lett, dinamikus, amilyen referenciára éppen mutat.

```
A a = new B(5);
```

A fenti példában az `a` változó statikus típusa `A`, dinamikus típusa `B` (megengedett, altípusos polimorfizmus, ld. Liskov-féle szubsztitúciós elv). A dinamikus típus leellenőrizhető az `instanceof` operátorral
09. Mi a különbség a túlterhelés és a felüldefiniálás között?
Felüldefiniálás Ha a *szignatúra* megegyezik
Túlterhelés Minden más esetben
A felüldefiniált függvények elé írjuk oda az `@Override` annotációt, és segít a compilernek kiszűrni az elgépelésből adódó problémákat fordítási időben. Ha hibát ad a fordító, valami nem stimmel (pl. nem egyezik a szignatúra).
Egy osztályon belül lehet több ugyanolyan nevű metódus is, ha a paraméterlistájuk különbözik – ezt hívjuk túlterhelésnek (overloading).
10. Hány féle adattípus dobható Java-ban és mi közöttük a különbség?
??
Checked Exceptions - kezelheto kivetelek, IO exception, etc. Altalaban a compiler is probal “kotelezni” kezelesre

Un-checked Exceptions - nem kezelheto, pl nullpointer
/*
Nem az Error és Exception-re gondol?
*/

11. Mi a különbség a változó láthatósága és élettartama között?
Hatókör: Ahol a változó használható

https://github.com/rlegendi/ELTE-javagyak/blob/master/04-oo-alapok/04-oo-alapok.md
Élettartam: Ahol a változó még *"él"*
12. Mikor és hogyan szabadulnak fel lokális változók JAVA-ban?
???kódblokk végén amiben definiáltuk őket???

Felszabadítással nem kell foglalkozni, azt megoldja a Garbage Collector (`finalize`). Memória: dinamikus/statikus/stack (utóbbihoz nem fértek hozzá), automatikusan felügyelt (_eden_, stb.), `System.gc()`, `finalize()`. Az aktuális értékek lekérdezhetők a`Runtime` osztály metódusaival.

// Nem lehet, hogy itt arra (is) gondoltak, hogy amint megszűnik a hatóköre? Mert hogy az egyben egy lokális változó élettartama is (ha úgy tetszik, az adott blokk ~után)

13. Milyen direkt memóriakezelési eszközök vannak JAVA-ban?
a sun.misc.Unsafe osztály memóriakezelő metódusai?

14. Milyen nyelvi elemek terhelhetők túl JAVA-ban?
?**Túlterhelés** (overloading): lehet több azonos nevű metódus, melyek a paraméterezésben és/vagy a visszatérési érték típusában térhetnek el egymástól.
Pl. float max(float a, float b) illetve int max(int a, int b)

15. Hogy jelenik meg a csomag-hierarchia a file rendszeren?
Almappákba szervezve: hu.elte.programom => hu/elte/programom

16. Mi a különbség az objektum és az osztály közt?
Lásd: 49. kérdés

17. Mit nevezünk példányosításnak?
Amikor egy osztályból létrehozunk egy új példányt a **new** kulcsszóval.
SajatClass osztaly = new SajatClass();

Peldanyositas: a new kulcsszoval hozunk letre egy pelanyt egy adott osztalybol. A new kulcsszo utan az osztaly egy konstruktorat kell megadni, utana zarojelben a konstruktor parametereit.
File f = new File("foo.txt");

18. Milyen belső osztályok vannak JAVA-ban?

Osztályok definiálhatók azonos forrásállományban (de csak egy `public` lehet), másik osztályban (belső osztályok), függvényen belül (lokális osztályok). Például:

```
package oo.basics;

class A { ... }
public class B {
    ...
    class C { ... }
    void f() {
        class D { ... }
        ...
    }
}
```

Beágyazott osztályok lehetnek `static`-ok, ekkor nincs szükség a befoglaló osztály egy példányára (így a példányváltozókhoz sem férnek hozzá). Ha egy osztály `final`, nem származtatható belőle újabb osztály.
Mikor melyiket használjuk?

- Ha szükség van egy adott osztály tagjainak elérésére, akkor beágyazott osztályt
- Ha úgy logikusabb, akkor beágyazott statikus osztály
- Ha más osztályban nincs rá szükség, de közvetlen nem függ össze a reprezentációval, akkor a `public` osztállyal azonos forrásállományba
- Ha általános osztályt készítünk, akkor saját fordítási egységbe

19. Mi a finally blokk?

try-catch utasításpár esetén a finally blokk minden esetben lefut (hacsak az előtte lévő blokkokban meg nem szakítjuk a program futását).
Ami mindenképp lefut???

A finally blokkot lényegében arra használjuk, hogy ha hiba történt a programban, akkor mégis helyesen fusson tovább. Akkor fut le, amikor a try-ból kilép a program.

Nem éppen .. Finally blokkot arra használunk, hogy a try után (akár hiba volt, akár nem) lefuttassunk néhány parancsot.

20. Mi a különbség az == és .equals() között?

== primitív, gyors... a referenciákat hasonlítja össze, nem az értékeket
.equals() lassabb, de... [== nem értelmezett String-re]

== értelmezett Stringre, viszont az csak egy általános cím szerinti összehasonlítás. Egyszerű típusokra általában jól működik, viszont objektumokra az Object osztályból örökölt equals függvényét kell használnunk.

== egyenlőséget hasonlít össze (azaz ugyanarról a dologról beszélünk)
.equals() ekvivalenciát hasonlít (azaz ekvivalens a kettő, ugyanúgy viselkedik, ugyan azt csinálja)

21. Sorolj fel minél több Object-ből örökölt metódust!

- clone()
- equals()
- finalize()
- getClass()
- hashCode()
- notify()
- notifyAll()
- toString()
- wait()
- wait(long timeout)
- wait(long timeout, int nanos)

22. Melyik metódusok definiálhatóak felül?

~ami nem final
Alap esetben az összes függvény virtual, kivéve, ha megjelöljük, hogy gyerek osztályok számára nem-overrideolhatóvá szeretnénk tenni (final kulcsszó). (Egy A osztály private függvénye felüldefiniálható lenne a gyerek osztályban, azonban a láthatóság miatt ezt nem tehetjük meg.)

23. Milyen eszköz van JAVA-ban konstruktor nélküli objektum létrehozásra?

Factory metódusok, osztályok lásd: BorderFactory

24. Mi a különbség az ArrayList és a LinkedList között?

Az ArrayList belsőleg egy tömb által van megvalósítva, a LinkedList elemei pedig önálló objektumok (a tárban). Mindkettő implementálja a List interface-t, így eszköztáruk ezen része megegyező, ugyanakkor belsőleg eltérő megvalósítású. Mivel a tömb konstans méretű, ezért az ArrayList elemének törlésekor mindig másolatot kell készítenie a tömbről, LinkedList-nél erre nincs szükség, ott csak néhány referenciát módosít. Viszont amíg az ArrayList elemeinek indexek általi elérése azonnali, addig LinkedList esetében ez csak közvetett módon,

szekvenciális bejárással lehetséges, ami (nagy elemszámnál) lassú.

25. Mi futtatja a JAVA fordító által generált tárgykódot?
//A Java Virtuális Gép azaz a JVM, ami a JRE (Java Futtatókörnyezet) része.

26. Mi az a bytecode?
Az a ”lefordított” bináris állomány amit a JAVA virtuális gép futtat.

27. Mikor hagyható el egy generikus típusparaméter megadása?
?? ha kiderül a meghívás más paramétereiből ??

28. Mi a különbség a definíció és a deklaráció között?
???definíciókor megadjuk a típust és nevet
???deklarációkor megadjuk az előtte definiálthoz az adatokat/műveleteket
//fordítva

29. Milyen lehetőség van JAVA-ban a típussal való paraméterezhetőségre?
Generic: Típussal paraméterezhetőség. Type erasure: csak fordítási időben ismert a típusinformáció, utána automatikusan törli a fordító, bájtkódból nem szerezhető vissza - nem C++ template-ek, nem generálódik fordítási időben új típus, nincs template metaprogramozás.

30. Mi JAVA-ban a destruktor szintaxisa és hogyan hívódik?
Destruktorok nincsenek, mert "szemétgyűjtés" van.
Tudomást szerezhetünk egy objektum megsemmisítéséről: finalize metódust kell megírni (Objectben van definiálva)
protected void finalize()
throws Throwable {...}
Pontosan nem definiált, hogy mikor hívódik meg: ami biztos, hogy a tárterület újrafelhasználása előtt.

31. Mi a különbség közöttük: forráskód, bytecode, gépi kód?
forráskód az, amit JAVA nyelven megírsz,
bytekód az, amit amit a fordító le generál magának
gépi kód az, amit a virtuális gép generál.

32. Milyen paraméter-átadási módok vannak a JAVA-ban?
Referencia szerinti, (érték szerinti, cím szerinti)

33. Írj egy szintaktikailag és szemantikailag helyes "Hello World" programot JAVA nyelven!

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println(“Hello world!”);
```

```
        }
}
```

34. Írd le mire való a super és az extends kulcsszavak generikus paraméterek megszorításánál!

Amikor tudjuk, hogy adott helyen csak adott osztály leszármazottai szerepelhetnek, első (rossz) megközelítés:

```
abstract class Super {}
class Sub1 extends Super {}
class Sub2 extends Super {}
...
void func(List<Super> l) {...} // Rossz!
```

Probléma: `func()` csak `List<Super>` paraméterrel hívható meg, `List<Sub1>`, `List<Sub2>` nem lehet paramétere (nem altípus). Megoldás: *bounded wildcard*:

```
void func(List<? extends Super> l) {...}
```

Belepakolni ugyanúgy nem tudunk, mint a `?` esetén, azaz erre fordítási hibát kapunk:

```
void func(List<? extends Super> l) {
    l.add(new Sub1()); // reccs
}
```

Felfelé is megköthető a wildcard a `<? super T>` jelöléssel.

https://github.com/rlegendi/ELTE-javagyak/blob/master/10-generics/10-generics.md

35. Magyarázd meg a type erasure kifejezést! Hol van JAVA-ban?
29-es kérdés?

36. Mennyi tárgykódot készít a JAVA fordító egy generikus osztályból?
egyet.

37. Mit jelent a változó-elfedés?
Egy függvényben definiált X nevű változó elfedi a függvényen kívül definiált másik ugyanilyen nevű változót, és csak a függvényen belüli változót tudjuk kezelni.

38. Mire való az @Override annotáció? Milyen hibáktól óvja meg a programozót?
Akkor használjuk mikor kimondottan felül szeretnénk írni egy adott metódust, és ha eltérünk a gyári fv. definíciótól, akkor compiler figyelmeztet, hogy mi felül szeretnénk írni, de valószínűleg nem úgy sikerült ahogy terveztük. (azaz pl más a paraméterezése, vagy a visszatérési értéke)

39. Írd le az equals függvény pontos szignatúráját! Mi a jellemző hiba equals implementálásakor? Mi óvja meg ettől a programozót?

public boolean equals(Object obj)

Az Object az az osztály, amiből minden másik osztály alapértelmezés szerint származik. Az equals esetén egy ilyen általános Object típusú objektumot kapunk, amit az adott típusra kell castolni ahhoz, hogy bárminemű (értelmes) összehasonlítást végezhessünk.

MyCalss o = (MyClass)obj;

Ennek az a veszélye, ha az obj nem MyClass típusú, akkor a program hibás működését kapjuk.

Ennek kiküszöbölésére lehet használni pl. az obj instanceof MyClass kifejezést, így egy feltételhez kötk, hogy el szeretnénk-e végezni az összehasonlítást. (Például ha A és B típusú objektumokat szeretnénk összehasonlítani, akkor biztosan nem egyezhetnek meg, mert különböző típusúak)

40. Sorold fel milyen vezérlési szerkezetek vannak a JAVA nyelvben! Írj mindegyikre egy-egy példát is!

A szekvencia egy szinte magától értetődő vezérlési elv, ahogy mi az írást, úgy a gép a programjainkat sorról-sorra olvassa. Azaz az adott sorban levő utasítás végrehajtása után (amennyiben az utasítás mást nem eredményezett) a következő sorba levő utasítás végrehajtása következik.

Szelekció: A programjaink adott pontjain, ahogy a való életben döntést kell hoznunk, hogyan tovább? (Vigyek-e esernyőt? Esik az eső? Ha …, ha nem …) E döntést felfoghatjuk úgy, mint egy kérdést. A kérdés kétféle lehet: eldöntendő(igen vagy nem?), vagy kiválasztó (Mennyi?), ennek megfelelően a program kettő, vagy több ágra ágazhat.

Iteráció: Az is gyakran elıfordul, hogy a programunkban egy, vagy több utasítást többször kell végrehajtani. Ekkor ciklikusan ismétlésbe kezd a program, míg egy adott feltétel, azt szükségessé teszi.

Értékadások, blokk, Elágazás(egyszerű és összetett), Ciklus(elől-, hátultesztelős, léptetős), ciklusmegszakítás, kilépés az alprogramból

41. Mik azok a literálok? Milyen literálok vannak a JAVA nyelvben?

A literál alatt bármely számot, szöveget vagy értéket értünk amely valamilyen értéket reprezentál.

http://www.roseindia.net/java/language/java-literals.shtml

42. Írd le mi a csellengő else probléma! Írj rá példát is!

6.3 => http://java2.uw.hu/06_elagazas.html

43. Milyen nem strukturált utasítások vannak a JAVA nyelvben?

- Címkézhető break és continue

- return

44. Van e goto a JAVA nyelvben?
Keywordkent letezik, de nincs hasznalva.
Helyette cimkézhető continue-break páros használható.

45. Mi a címkék szerepe? Írj egy példát a használatára?
// címkézett break, continue?
http://en.wikipedia.org/wiki/Java_syntax#Labels

46. Hogyan lehet JAVA-ban változó számú argumentumlistát használni? Mik az erre vonatkozó megszorítások?
Java 5.0-tól, c++-ban megszokott ... jelöléssel

```
public void write(String...  records) {
        for (String record: records)
                System.out.println(record);
}

write("line 1");
write("line 2", "line 3");
write("line 4", "line 5", "line 6", "line 7");
```

47. Írd le milyen kommentezési és dokumentálási lehetőségek vannak JAVA-ban?

```
// Egysoros
/* Többsoros1
 * Többsoros2
 */
```

Javadoc részére:

```
/**
*/

/** @param, @result, @see */
```

48. Milyen láthatóságokat ismersz? Melyik mire jó?
public: Bárki láthatja, pl leszármaztatás után is, vagy az adott csomagból bárki. (csomagon kívül is bárki!)
protected: csomagból,ős és gyermek osztályok láthatják.

package-private (más néven *default* - ha nem írsz semmi láthatóságot, ilyen lesz*)*: Az adott csomagon belül látható.
private: Senki nem láthatja csak az osztály maga.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

49. Mi a különbség az objektum és az osztály között?
Az osztály az objektum “terve”.
Az objektum az osztály egy példánya.

Osztaly: osztalyvaltozok es rajtuk ertelmezett metodusok
Objektum: Egy osztaly peldanya. Egy objektumnak van identitasa, ami egy referencia (egy pointer egy adott memoriacimre) es allapota. Az objektum allapota az adattagjainak az erteke. Egy osztalynak tetszoleges szamu peldanya lehet.

50. Mire jó a final kulcsszó? Változhat-e egy final referenciaval rendelkező objektum belső állapota?
konstans deklaráció elviekben nem, pont emiatt kapja a final kulcsszót. Használható classokon, methodokon, és változókon, és inner classokon.
// az objektum referenciája nem, de a belső állapota változhat // szóval elviekben, de :)

51. Milyen összetett típusok vannak JAVA-ban?
Összetett típusok: osztályok; értékeik: objektumok
Pl.: a String egy osztály; a String típusú objektumok a karakterláncok. A "szöveg" literál egy String típusú objektumot jelöl.
Összetett értéket csak objektummal lehet létrehozni: az egyetlen típuskonstrúkció.

52. Milyen lehetőség van a JAVA nyelvben objektumok bináris formátumba való mentésére és visszaolvasására?
Serializable : Objektumok "sorosítása", hogy túléljék a JVM-et: elmenthetők, Socketen átküldhetők, DB-ben tárolhatók, etc. Az objektum állapotát bájtokba toljuk ki.
Implementálni kell a java.io.Serializable interfészt (ún. marker interfész). Ilyenkor a default szerializáció minden adattagot kiment.

53. Mire való a Comparable inteface? Írd le pontosan mi a Comparable szerződése!
Valamilyen reláció renszerint dezhető osztály megvalósítására.

54. Hogyan lehet rendezést definiálni olyan objektumokon, amik nem implementáljak a Comparable interface-t?
Comparator interface megvalósításával:
Comparator: int compare(Object o1, Object o2)

55. Mi a különbség a konstruktor és a factory method között?
hogy nem kell megadni a visszatérési értékét, mert az adott.
A factory method egy osztály osztályszintű metódusa, amit példányosításra használ (ott megadják a visszatérési értéket)

56. Mikor kell meghívni az ősosztály konstruktorát? Miért?A konstruktor úgy példányosít,
A konstruktor nem örököltethető, de meghívható a szülőosztálybeli konstruktor a legelső sorban: super névvel
A super megelőzi az osztálydefinícióban szereplő példányváltozó inicializálásokat, a többi része a konstruktornak viszony csak utánuk jön

Mert az ősosztálynak nincs default (nullparaméteres) konstruktora, így ha nem kéne meghívni, olyan adattagok lehetnek elérhetőek, amelyeknek nem adtunk kezdőértéket.

57. Írd le mi az az explicit típuskényszerítés! Mikor használjuk?
Bővebb halmazba tartozó értékeket egy szűkebb halmazba tarto értékké konvertál: adatvesztő, nem biztonságos
pl. floatból int-et: Math osztály kerekítő függvényével: egészek szűkítése esetén felső bitek vesznek el //A kerekítőfv az biztos kényszerítés?

58. Írd le mi az a dinamikus kötés!
Mindig a dinamikus típus szerinti művelet hívódik meg
Futás közben választódik ki az a metódus ami végrehajtódik

59. Mit jelent az, hogy egy objektum immutable? Milyen immutable típusokat ismersz?
Az objektum állapota (pl.: belső adattagok) a létrehozás után nem változhat meg. Például: String, Integer // valamint az összes primitív
A nyelvben léteznek ún. immutable (változtathatatlan) objektumok, melyek értékeit csak készítéskor lehet beállítani. Ha egy ilyen objektumot szeretnénk módosítani, azt csak annak lemásolásával – új objektum létrehozásával lehet

60. Túlindexelhetőek-e JAVA-ban a tömbök? Mi történik pontosan túlindexelés eseten?
[valszeg nem] ArrayIndexOutOfBoundsException
Túlindexelhetőnek túlindexelhető, exceptiont (ArrayIndexOutOfBoundsException) kapunk

61. Van e JAVA-ban többdimenziós tömb? Mi a szintaxisa?

Ket dimenzios t□omb□ok olyan 1 dimenzios t□omb□ok amelyek 1 dimenzios t□omb□oket tartalmaznak.
E miatt inicializalasnal eleg az 1. dimenziot megadni.
Peldak:

```
int[][] arr1 = new int[5][4];
int[][] arr2 = new int[5][];
arr2[0] = new int[2];
arr2[1] = new int[3];
```

62. Mi van JAVA-ban többdimenziós tömb helyett? Mi a szintaxisa?

1 dimenziós tömbökből álló tömb.

63. Sorold fel a collections framework minél több típusát! Milyen közös műveleteik vannak?

Set, HashSet, TreeSet, LinkedHashSet
List, ArrayList, LinkedList
Map, HashMap, TreeMap, LinkedHashMap
Queue

sort(List)
binarySearch(List, Object)
reverse(List)
shuffle(List)
fill(List, Object)
copy(List dest, List src)
min(Collection)
max(Collection)
rotate(List list, int distance)
replaceAll(List list, Object oldVal, Object newVal)
indexOfSubList(List source, List target)
lastIndexOfSubList(List source, List target)
swap(List list, int, int)

http://docs.oracle.com/javase/1.4.2/docs/guide/collections/reference.html

64. Milyen eszközt ad a JAVA nyelv struktúrák transzparens bejárására?

Iterátor

65. Sorold fel a java.util.List interface minél több műveletét! Írd le melyik mire való!
http://docs.oracle.com/javase/1.4.2/docs/api/java/util/List.html

add()
addAll()
clear()
contains()
equals()
remove()
size()
get(int i)
isEmpty()

66. Sorold fel a java.util.Collections osztály minél több műveletét! Írd le melyik mire való!
http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Collection.html
/* nem Colletction interface-t, hanem a Collections osztályt kérdezi... http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Collections.html */
add(Object o)
addAll(Collection c)
clear()
contains (Object o)
containsAll(Collection c)
equals(Object o)
hashCode()
isEmpty()
iterator()
remove(Object o)
removeAll(Collection c)
retainAll(Collection c)
size()
toArray()
toArray(Onject[] a)

67. Sorold fel a java.util.Arrays osztály minél több műveletét! Írd le melyik mire való!

asList()
binarySearch()
copyOf()
copyOfRange()
deepEquals()
deepHashCode()
deepToString()
equals()

fill()
hashCode()
sort()
toString()

http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html

68. Mire jó az import kulcsszó JAVA-ban? Mit jelent a következő: import java.io.*;

•Arra használható, hogy ne kelljen kiírni bizonyos típusok teljes nevét (pl. java.util.Vector), csak a rövid nevét (Vector).
•Nem elég, hogy **`import java.*;`**
•Nem használható „félig teljes” név, pl. **`util.Vector`**

import java.io.*: minden típust (osztályt) importálunk egyszerre a java.io csomagból
.

69. Mire jó az import static JAVA-ban? Mit jelent a következő: import java.util.Collections.*;

•A Java 5-től kezdve nem csak típusok, hanem statikus tagok is importálhatók
–Statikus metódusok
–Statikus attribútumok
•import static utasítás
•Importálható egy utasítással
–egy név, vagy
–egy típus összes statikus tagja

70. Írj egy JAVA programot, ami rekurzívan kiszámolja n! értékét!

```
int fact(int n){
        int result;
        if(n==1) {
                return 1;
        }
        result = fact(n-1) * n;
        return result;
}
```

71. Írj JAVA programot, ami iteratívan kiszámolja n! értékét!

```
int factorial(int n) {
        int result = 1;
        for ( int i=1; i<=n; i++ ) {
                result *= i;
```

```
        }
        return result;
}
```

72. Mire jó a @Deprecated annotáció!

Olyan kodreszlet aminek a hasznalata nem ajanlott vagy van jobb alternativa.
A JAVA következő verziói már nem támogatják.

73. Fejtsd ki mi az előnye illetve a hátránya a kivételkezelésnek!

Kivétel: A programot az átlagos/normális végrehajtási menetétől eltérítő esemény (pl.: futási idejű hiba)
Előnyei: olvashatóság, karbantarthatóság, továbbfejleszthetőség, megbízhatóság, stabilitás
Hátrányai: ??? (plusz kódolás)
// A probléma valójában az, hogy romlik az olvashatóság, magas skálázásnál már valós sebességbeli különbséget okozhat, illetve ha átírod akkor az őt használó osztályban is változtatni kell ezeken ha ott kapod el, illetve az általános exceptionökkel gyakran elfeded a valós hibát, ezért kell figyelni, hogy a lehető legspecifikusabb hibát kapja el az ember, pl Hiába lehet az Exception nevűt, ha a kódom mondjuk tud IOExceptiont meg NumberFormatExceptiont akkor azt kell elkapni, különben fogalmam sem lesz melyik dobta. ***Így a tovább fejleszthetőség nem igazán előnye, szituációtól függhet.***
// még évelején mesélt a tanár egy olyan "hátrányról", hogy ezzel ki is lehet akasztani a jvm-et, ha az exception esetében finally blokkban valami "nagy marhaságot" írunk (pl elindítjuk újra a programot) mert a finally ág mindenképp lefut így outofmemoryexception sem lépteti ki a programot és "meghal az egész" (nem tudom ez mennyire helyt álló de ilyenről mesélt félév elején)

74. Hasonlítsd össze a kivételkezelést az "errorcode-os" hibakezeléssel! (előnyök, hátrányok)

Kivételkezeléssel több infóhoz juttathatjuk az usert és ezáltal magunkat is.
Nem kell figyelni, hogy azonos típusú de extremális elemet adjunk vissza (pl faktoriális számító programnál még jó return hibakód a -1, de természetes számok összeadására már nem)

75. Hogyan terjednek a kivételek JAVA-ban? Hogyan kezelhetjük őket? Írj rá példát is!

A hívási lánc mentén
        – A végrehajtási verem mentén
Ha egy m metódusban kivétel lép fel, akkor az azt meghívó metódusban is fellép, azon a ponton, ahol meghívtuk az m metódust
        – hacsak persze le nem kezeljük…
Egészen addig, amíg a main-ben is fel nem lép: ekkor leáll a program, és kiírja a kivételt
        – stack trace

76. Sorolj fel minél több Exceptiont és magyarázd el, hol használjuk őket!

OutOfMemoryError
NullpointerException
StackOverflowException
StackUnderflowException
ArrayIndexoutOfBoundsException
IllegalArgumentException
IOException
Exception