Programozási nyelvek 2

Szathmáry László
Debreceni Egyetem
Informatikai Kar

Öröklődés

- kvíz

(utolsó módosítás: 2022. nov. 11.)

Q1

Legyen adott két osztály, *A* és *B*. Hogyan tudjuk jelölni, hogy *A* alosztálya a *B* osztálynak?

a) `class A inherits B { … }`
b) `class A extends B { … }`
c) `class A : B { … }`
d) `class A(B) { … }`
e) `class A implements B { … }`

A1

Legyen adott két osztály, *A* és *B*. Hogyan tudjuk jelölni, hogy *A* alosztálya a *B* osztálynak?

a) `class A inherits B { … }`
b) `class A extends B { … }`
c) `class A : B { … }`
d) `class A(B) { … }`
e) `class A implements B { … }`

Q2

Mi lesz az alábbi kód kimenete? Válaszát indokolja!

```
Point3D p1 = new Point3D(1, 3, 7);
Point3D p2 = new Point3D(1, 3, 7);

System.out.println(p1.equals(p2));
```

A2

Mi lesz az alábbi kód kimenete? Válaszát indokolja!

```
Point3D p1 = new Point3D(1, 3, 7);
Point3D p2 = new Point3D(1, 3, 7);

System.out.println(p1.equals(p2));
```

---

false

Ha a `Point3D` osztályban nem írtuk felül az `equals()` metódust, akkor az `Object` osztály `equals()` metódusa hívódik meg, ami a referenciákat az értékük alapján hasonlítja össze. Vagyis azt nézi meg, hogy a két referencia ugyanoda mutat-e. Mivel p1 és p2 két különböző memóriaterületre mutat, ezért az összehasonlítás eredménye hamis lesz.

Q3

Mit ad vissza az `Object` osztály `hashCode()` metódusa?

Mi a visszaadott érték típusa?

A3

Mit ad vissza az `Object` osztály `hashCode()` metódusa?

Mi a visszaadott érték típusa?

Egy egész (`int`) típusú hash értéket ad vissza, amit az objektum memóriacíme alapján számít ki.

Q4

Mit értünk alapértelmezett konstruktor alatt (*default constructor*)?

A4

Mit értünk alapértelmezett konstruktor alatt (*default constructor*)?

A paraméter nélküli konstruktort. Ha nem adunk meg egy ilyen konstruktort, akkor a Java fordító automatikusan létrehoz az osztály számára egy ilyet.

Q5

Hogyan tudunk létrehozni egy konstruktort a `Seller` osztályban?

a) `public Seller(String name) { … }`
b) `public void Seller(String name) { … }`
c) `public Constructor(String name) { … }`
d) `public void Constructor(String name) { … }`

A5

Hogyan tudunk létrehozni egy konstruktort a `Seller` osztályban?

a) `public Seller(String name) { … }`
b) `public void Seller(String name) { … }`
c) `public Constructor(String name) { … }`
d) `public void Constructor(String name) { … }`

Q6

Mire szolgál a `super` kulcsszó?

A6

Mire szolgál a `super` kulcsszó?

A segítségével hivatkozni tudunk a szülőosztály attribútumaira és metódusaira. A szülőosztály konstruktorát / konstruktorait is ennek a segítségével tudjuk meghívni.

(A `this` kulcsszó ezzel szemben egy olyan referencia, ami az aktuális objektumra mutat).

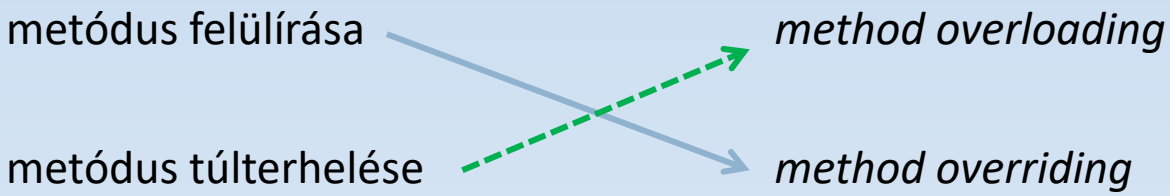
Q7

Párosítsa össze a megfelelő fogalmakat!

metódus felülírása	*method overloading*
metódus túlterhelése	*method overriding*

A7

Párosítsa össze a megfelelő fogalmakat!



Q8

Mit értünk egy metódus felülírása (*overriding*) alatt?

Miben más ez, mint egy metódus túlterhelése (*overloading*)?

A8

Mit értünk egy metódus felülírása (*overriding*) alatt?

Miben más ez, mint egy metódus túlterhelése (*overloading*)?

---

metódus felülírása:

Az öröklődéshez kapcsolódó fogalom. Amikor az alosztályban nem vagyunk elégedettek az örökölt metódus implementációjával, s lecseréljük az implementációt. Ilyenkor a metódus szignatúrája változatlan marad!

metódus túlterhelése:

Amikor több metódusnak is azonos a neve. A metódusok formális paraméterlistáinak eltérőnek kell lenniük!

Q9

Egy metódus felülírása (*overriding*) esetén mi haszna van a `@Override` annotáció alkalmazásának?

A9

Egy metódus felülírása (*overriding*) esetén mi haszna van a `@Override` annotáció alkalmazásának?

A `@Override` használata esetén a fordító leellenőrzi, hogy az adott metódus tényleg szerepel-e a szülőosztályban, ill. a két metódus szignatúrája megegyezik-e.

Akkor is hasznos, ha esetleg a szülőosztályból a későbbiekben töröljük ezt a metódust. Ekkor a fordító a gyermekosztálynál hibát jelez.

Q10

A `Seller` osztály a `User` osztály leszármazottja. Az alábbi metódust meg lehet hívni egy `Seller` típusú objektummal? Miért?

```
public void print(User u) { … }
```

A10

A `Seller` osztály a User osztály leszármazottja. Az alábbi metódust meg lehet hívni egy `Seller` típusú objektummal? Miért?

```
public void print(User u) { … }
```

Igen. Egy `Seller` típusú objektum egyúttal példánya a `User` osztálynak is. A metódus meghívásakor automatikus upcasting történik (egy gyermekosztály példányát a szülőosztály típusára konvertáljuk).
A `print`() metódusban ekkor csak a `User` osztályban megadott metódusok hívhatók meg. Ha a `Seller` osztály metódusaihoz is hozzá szeretnénk férni, akkor az u objektumot downcast-olni kell `Seller` típusúra.

Q11

Mire szolgál az `instanceof` operátor?

A11

Mire szolgál az `instanceof` operátor?

A segítségével le tudjuk kérdezni, hogy egy adott objektum példánya-e egy adott osztálynak. Ezen operátor használatával elkerülhető a `ClassCastException` hiba.

Az öröklődés miatt egy alosztály példánya egyúttal az összes szülőosztálynak is példánya.

Q12

Mikor használunk absztrakt (*abstract*) osztályokat?

A12

Mikor használunk absztrakt (*abstract*) osztályokat?

Amikor egy osztályt csak azért hozunk létre, hogy a gyermekosztályai megörököljék az attribútumait / metódusait. Maga az osztály túl absztrakt ahhoz, hogy példányosítsuk.

Egy absztrakt osztály nem példányosítható.

Q13

Létre lehet-e hozni egy absztrakt osztályt úgy, hogy nincs neki egyetlen absztrakt metódusa sem?

A13

Létre lehet-e hozni egy absztrakt osztályt úgy, hogy nincs neki egyetlen absztrakt metódusa sem?

Igen.

Q14

Egy nem-absztrakt osztályban meg lehet-e adni egy absztrakt metódust?

A14

Egy nem-absztrakt osztályban meg lehet adni egy absztrakt metódust?

Nem. Ha egy metódust absztrakttá teszünk, akkor az osztálynak is absztraktnak kell lennie.

Q15

Mi az absztrakt metódus?

A15

Mi az absztrakt metódus?

Egy absztrakt metódusnak csak a deklarációját (szignatúráját) lehet megadni.
Egy absztrakt metódusnak nincs implementációja.

Q16

Mi a final osztály?

Mi a final metódus?

A16

Mi a final osztály?

Mi a final metódus?

---

final osztály:

Nem terjeszthető ki.

final metódus:

Nem lehet felülírni. Vagyis a gyermekosztály nem cserélheti le az implementációját.

Q17

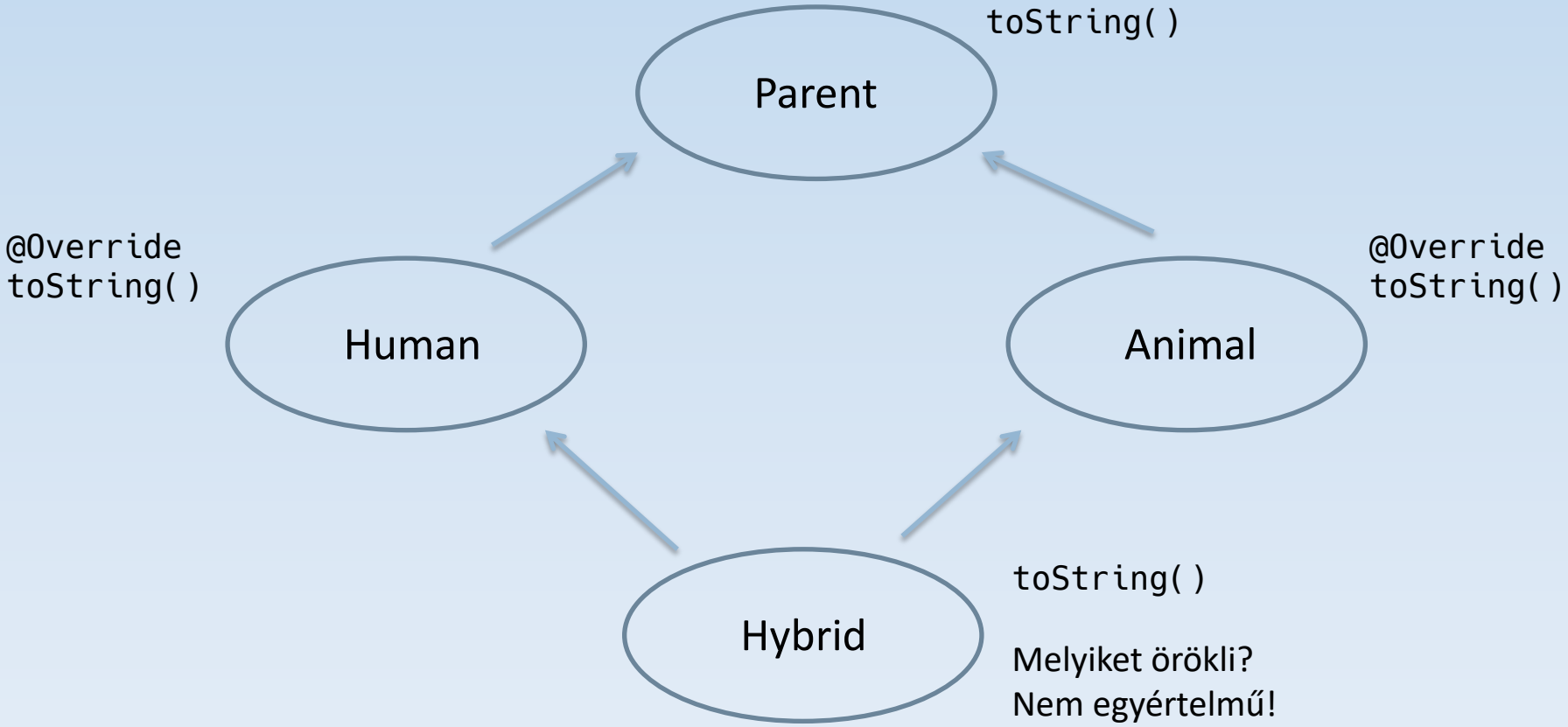
Mi a gyémánt probléma (*diamond problem*)?

A17

Mi a gyémánt probléma (*diamond problem*)?

Többszörös öröklődés esetén jöhet elő. Legyen *A* egy osztály, aminek van egy metódusa (pl. `toString`()). A *B* és a *C* osztályok az *A* gyermekei, s mindkettő felülírja az *A*-tól örökölt metódust.
D terjessze ki *B*-t és *C*-t is. *D* a metódus melyik implementációját örökli meg? A *B* által felülírtat, vagy a *C* által felülírtat? Nem egyértelmű ☹

A17


toString()
Parent
@Override
toString()
Human
Animal
@Override
toString()
Hybrid
toString()
Melyiket örökli?
Nem egyértelmű!

Q18

Java-ban egyszeres vagy többszörös öröklődés van?

A18

Java-ban egyszeres vagy többszörös öröklődés van?

Egyszeres.