Objektovo-orientované programovanie **B**

Ing. Valentino Vranić, PhD., ÚISI FIIT STU

Skúška — 23. máj 2007

Priezvisko:	
Meno:	

1 b	
2 b	
3 b	

Skúška trvá 120 minút.

V otázkach 1–16 je len jedna možnosť správna. Vyznačte svoju odpoveď krížikom do veľkej tabuľky (malú tabuľku nevypĺňajte). Hodnotia sa len odpovede vyznačené v tabuľke.

V prípade opravy jasne vyznačte ktorú odpoveď vyberáte. Každá správna odpoveď má hodnotu vyznačenú v otázke. Nesprávna odpoveď, vyznačenie viac odpovedí alebo nejednoznačné vyznačenie má hodnotu 0 bodov. Postup riešenia sa pre otázky 1–16 nehodnotí.

Odpovede na otázky 17 a 18 píšte na prídavný list. Na ňom tiež uveďte svoje priezvisko a meno.

	a	b	c	d	e
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

1. (1 b) Objekt v Jave predstavuje

(a) triedu
(b) typ
(c) inštanciu triedy alebo rozhrania
(d) modul
(e) inštanciu triedy

2. (1 b) V triede, ktorá implementuje rozhranie, je možné zadefinovať

(a) len ďalšie polia
(b) len ďalšie metódy
(c) len metódy deklarované v rozhraní
(d) ľubovoľné ďalšie polia a metódy
(e) len metódy nedeklarované v rozhraní

3. (1 b) Pole triedy, ktorému predchádza kľúčové slovo **protected**

(a) je dostupné len v rámci jednej nite
(b) je chránené pred zápisom
(c) je dostupné len v danej hierarchii tried
(d) sa nezapíše do súboru pri serializácii objektu
(e) je dostupné len v danej triede

4. (1 b) Ak trieda v jazyku C++ obsahuje čisto virtuálne funkcie, potom

(a) nemôže mať inštancie
(b) sa od nej dá dediť len virtuálne
(c) nemôže dediť od triedy, ktorá ich neobsahuje
(d) môže dediť od inej triedy len virtuálne
(e) nepodporuje polymorfizmus

5. (2 b) Program v jazyku C++ obsahuje triedu Lopta s implementovanou metódou skoc(). Daný je nasledujúci kód:

```
Lopta o;
o.skoc();
```

Tento kód je v jazyku C++

(a) korektný, ale jedine ak je trieda Lopta virtuálna
(b) nekorektný
(c) korektný, ale jedine ak je metóda skoc() virtuálna
(d) korektný
(e) korektný, ale jedine ak je metóda skoc() statická

6. (2 b) Metóda f() triedy A vyhadzuje výnimku MyException. Daná je trieda B:

```
class B {
   void m() { new A().f(); }
}
```

Metóda m() triedy B

(a) musí deklarovať že vyhadzuje výnimku typu MyException
(b) musí vyhadzovať výnimku typu MyException
(c) musí ošetrovať výnimku typu MyException
(d) je korektná
(e) musí deklarovať alebo ošetrovať výnimku typu MyException

7. (2 b) V aspektovo-orientovanej implementácii vzoru Observer

(a) oddeľuje sa aplikačná logika od rozhrania aplikácie
(b) vyčleňuje sa do aspektu logika vzoru, ktorá je predovšetkým v časti Subject
(c) vyčleňuje sa do aspektu logika vzoru, ktorá je predovšetkým v časti Observer
(d) vzniká tretia časť vzoru označovaná ako Mediator medzi časťami Subjec a Observer
(e) zaniká potreba implementovať Subject

8. (2 b) Vzťah «use» v jazyku UML v smere od triedy k rozhraniu znamená, že

(a) trieda vytvára inštanciu rozhrania
(b) trieda ovplyvňuje rozhranie
(c) trieda volá všetky metódy rozhrania
(d) trieda volá niektorú z metód rozhrania
(e) trieda implementuje metódy predpísané rozhraním

9. (2 b) Jeden z rozdielov medzi abstraktnou triedou a rozhraním v Jave je ten, že

(a) abstraktná trieda môže dediť od rozhrania, ale nie naopak
(b) rozhranie môže dediť od abstraktnej triedy, ale nie naopak
(c) konkrétne triedy môžu dediť od abstraktnej triedy, ale nie od rozhrania
(d) je možné vytvárať inštancie rozhraní, ale nie aj abstraktných tried
(e) je možné vytvárať inštancie abstraktných tried, ale nie aj rozhraní

10. (2 b) V jazyku AspectJ je pomocou videnia (advice) možné

(a) pridať novú metódu do aspektu
(b) zmeniť vykonávanie metódy
(c) definovať nové závislosti medzi triedami
(d) pridať novú metódu do triedy
(e) definovať bod spájania

11. (3 b) Vytvorili ste tlačidlo t ako komponent rámca Swing a zviditeľneli ste okno, ktoré ho obsahuje. Potrebujete zmeniť označenie (label) tlačidla na text Abc. Urobíte to volaním

(a) SwingUtilities.run(**new** Runnable() {
public void invokeLater() {t.changeText("Abc");}});
(b) t.changeText("Abc")

(c) SwingUtilities.invokeLater(t.changeText("Abc"));
(d) SwingUtilities.invokeLater(**new** Runnable() {
public void run() {t.changeText("Abc");}});
(e) (**new** Runnable(SwingUtilities.invokeLater(
t.changeText("Abc");}}))).run();

12. (3 b) Daný je kód v Jave na obr. 1. Vykonaním týchto príkazov:

```
A o = new X();
o.m();
((X)o).m();
((A)o).m();
```

(a) pole i nadobudne hodnotu 1
(b) pole i nadobudne hodnotu -3
(c) pole i nadobudne hodnotu 3
(d) pole i nadobudne hodnotu -1
(e) vznikne chyba v poslednom riadku

```
abstract class A {
    int i = 0;
    public abstract void m();
}

class X extends A {
    public void m() { i--; }
}

class Y extends X {
    public void m() { i++; }
}
```

Obrázok 1: Kód pre otázky 12 a 14.

13. (3 b) Návrhový vzor Observer je vo vzore Model-View-Controller využitý vo vzťahu

(a) Model-View
(b) tried vo vnútri časti View
(c) tried vo vnútri časti Controller
(d) View-Controller
(e) Model-Controller

14. (3 b) Ku kódu v Jave na obr. 1 je daná nasledujúca trieda:

```
class M {
    static void m(Class<? extends A> T, A... o) {
        for (A e : o)
            if (T.isInstance(e))
                System.out.print("a");
    }

    public static void main(String... args) {
        m(X.class, new A[]{new X(), new Y()});
    }
}
```

Jej vykonaním

(a) vypíše sa aaa
(b) vznikne výnimka
(c) nevypíše sa nič
(d) vypíše sa aa
(e) vypíše sa a

15. (3 b) Daný je nasledujúci kód v Jave:

```
**1** B {
    **2**
}

public class A {
    **3** f() {
        return new **3**() {
            public void m() { . . . }
        };
    }
    public void g() {
        new A().f().m();
    }
}
```

Ktoré fragmenty kódu treba v tomto programe doplniť, aby bol korektný?

(a) `**1**`: **interface** `**2**`: **void** m(); `**3**`: B
(b) `**1**`: **class** `**2**`: **void** f(); `**3**`: B
(c) `**1**`: **interface** `**2**`: **void** f(); `**3**`: B
(d) `**1**`: **class** `**2**`: **void** m(); `**3**`: A
(e) `**1**`: **class** `**2**`: **void** m(); `**3**`: B

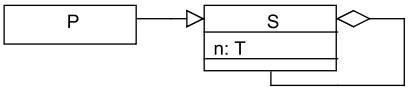
16. (3 b) Daný je nasledujúci kód v Jave:

```
if (o.class == "Kruh")
    ((Kruh)o).nakresli();
else if (o.class == "Stvorec")
    ((Stvorec)o).nakresli();
else
    ;
```

Tento kód porušuje

(a) pravidlá dedenia
(b) Liskovej princíp substitúcie
(c) pravidlá polymorfizmu
(d) princíp zapuzdrenia
(e) princíp otvorenosti a uzavretosti

17. (4 b) Prepíšte diagram na obr. 2 do kódu v Jave.



Obrázok 2: Diagram pre otázku 17.

18. (12 b) Aplikácia na modelovanie umožňuje vytváranie a spájanie prvkov. Pri vytvorení prvku sa nastaví jeho celočíselná ohodnotenie a počet prvkov, s ktorým ho možno spojiť. Spojenie sa vždy týka dvoch prvkov, je usmernené a pozná ho len zdrojový prvok.

Každý prvok je jedného z troch druhov: Adder, Subtracter alebo Multiplier. Adder zvýši ohodnotenie prvku, s ktorým sa spája, o svoje ohodnotenie. Subtracter zníži ohodnotenie prvku, s ktorým sa spája, o svoje ohodnotenie. Multiplier vynásobí ohodnotenie prvku, s ktorým sa spája, svojim ohodnotením.

Napíšte relevantný kód v Jave, ktorý zodpovedá týmto požiadavkám, využívajúc pritom mechanizmy objektovo-orientovaného programovania v maximálnej miere. Kód vysvetlite.

Objektovo-orientované programovanie **B**
Ing. Valentino Vranić, PhD., ÚISI FIIT STU
Skúška — 23. máj 2007

1 e

2 d

3 c

4 a

—

5 d

6 e

7 b

8 d

9 a

10 b

—

11 d

12 b

13 a

14 d

15 a

16 e

50