Objektovo-orientované programovanie 2022/23
doc. Ing. Valentino Vranić, PhD., ÚISI FIIT STU
Skúška – opravný termín – 19. jún 2023

(vyplňte tlačeným písmom)
Priezvisko:

Meno:

1 b	
2 b	
3 b	

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Skúška trvá 70 minút.
Odpovede na otázky 1–12 vpíšte do tabuľky. Pri týchto otázkach sa hodnotia len odpovede v tabuľke (bez postupu). Odpoveď musí byť jednoznačná a čitateľná, inak má hodnotu 0 bodov.
V otázkach s ponúknutými odpoveďami je len jedna možnosť správna – do tabuľky vpíšte len písmeno, ktorým je označená odpoveď, ktorú vyberáte.
Odpoveď na otázku 13 píšte výlučne na list, na ktorom sa nachádza jej znenie.
Poškodený list nebude uznaný.

1. (1 b) V jazyku C# zástupca (delegate) reprezentuje
(a) funkciu zodpovedajúcich typov parametrov a návratovej hodnoty
(b) ukazovateľ na objekt
(c) hocijakú vlastnosť
(d) vlastnosť zodpovedajúceho typu
(e) hocijakú funkciu

2. (1 b) Serializácia v Jave slúži na
(a) agregáciu údajov
(b) uchovanie údajov
(c) uloženie zdrojového kódu
(d) ochranu údajov
(e) synchronizáciu údajov

3. (1 b) V jazyku C++ prekonávanie sa použitím špeciálneho kľúčového slova
(a) môže v prípade, že nie je potrebné, deaktivovať
(b) musí v prípade potreby aktivovať, a následne sa môže deaktivovať
(c) musí v prípade, že nie je potrebné, deaktivovať
(d) musí v prípade potreby aktivovať, a následne sa musí aj deaktivovať
(e) musí v prípade potreby aktivovať

4. (1 b) V jazyku C++ sa deštruktor
(a) musí uviesť v triedach, ktoré obsahujú virtuálne metódy
(b) musí uviesť v triedach, v ktorých sa pri tvorbe ich objektov dynamicky alokuje pamäť
(c) neuvádza nikdy
(d) musí uviesť v triedach, ktoré sú odvodené od viacerých iných tried
(e) musí uviesť v každej triede

5. (1 b) Určitou nevýhodou aspektov v jazyku AspectJ je, že
(a) nedokážu pridať nové prvky do tried
(b) je možné ich mať iba v malom počte
(c) nedokážu ovplyvniť používateľské rozhranie
(d) nedokážu nahradiť jestvujúce metódy v triedach
(e) v ovplyvnenom kóde táto skutočnosť nie je viditeľná

6. (1 b) Java podporuje perzistenciu prostredníctvom
(a) serializácie
(b) modularizácie
(c) agregácie
(d) radializácie
(e) synchronizácie

7. (2 b) Daný je nasledujúci program v Jave:

```
class C implements Runnable {
    M m;
    public C(M m) {
        this.m = m;
    }
    public synchronized void run() {
        for (int i = 0; i < 100000; i++) {
            m.c();
            m.a();
        }
    }
}

public class M {
    private char x = 'a', y = 'a';

    public synchronized void a() {
            if (x != y)
                System.out.println("a");
            x = 'a';
            y = 'a';
    }

    public synchronized void b() {
        synchronized(this) {
            if (x != y)
                System.out.println("b");
            x = 'b';
            y = 'b';
        }
    }

    public synchronized void c() {
        if (x != y)
            System.out.println("c");
         x = 'c';
         y = 'c';
    }

    public static synchronized void main(String[] args) {
        M m = new M();
        new Thread(new C(m)).start();
        new Thread(new C(m)).start();
    }
}
```

Pri ktorých metódach je možné odstrániť modifikátor **synchronized**, aby stále bolo zaručené, že sa nič nikdy nevypíše (uveďte ich v zápise: `Trieda.metóda()`)?

8. (2 b) V architektonickom vzore MVC je vzťah Model–View vhodné realizovať

(a) návrhovým vzorom Strategy
(b) návrhovým vzorom Observer
(c) bez použitia akéhokoľvek návrhového vzoru
(d) návrhovým vzorom Composite
(e) návrhovým vzorom Visitor

9. (2 b) Daná je trieda v Jave:

```
class A {
   void z() throws Q {
      ...
   }
   void m() throws R, Q {
      try {
         z();
      } catch (Q q) {
         ...
      }
      ...
   }
}
```

Z tohto kódu možno usúdiť, že metóda m()

(a) môže vyhodiť výnimku typu Q
(b) vždy vyhadzuje výnimku typu Q
(c) môže vyhodiť výnimku typu R alebo Q
(d) vždy vyhadzuje výnimku typu R
(e) môže vyhodiť výnimku typu R

10. (2 b) Daná je časť kódu hry v Jave:

```
class Ovladanie {
   public static void main() {
      ...

      if (tah == 'a') { // zautoc na nepriatela
         if (player.hasSword()) {
            enemy.decreaseEnergy(10);
         } else {
            player.decreaseEnergy(10);
         }
      } else if...
      ...
   }
}
```

Z hľadiska flexibility objektovo-orientovaného návrhu by bolo najdôležitejšie

(a) vyčleniť kód podmienený ťahom `a` do metódy nezávislej od ovládania
(b) vyčleniť celé ovládanie hry do nestatickej metódy
(c) vytvoriť grafické používateľské rozhranie
(d) namiesto príkazu **if** použiť príkaz **case–switch**
(e) zmeniť metódu decreaseEnergy() na statickú

11. (3 b) Čo sa vypíše po spustení nasledujúceho programu v Jave?

```
class C {
   public void f() {
      System.out.print("C");
   }
}
class D extends C {
   public void f() {
     super.f();
      System.out.print("D");
   }
}
class E extends D {
   public void f() {
     super.f();
     System.out.print("E");
   }
}
class F extends E {
   public void f() {
      System.out.print("F");
   }
}
public class M {
   public static void main(String[] args) {
      F o1 = new F();
      D o2 = new E();
      C o3 = new F();
      E o4 = new E();
      C o5 = new D();

      ((E) o1).f();
      System.out.print(" ");

      ((E) o2).f();
      System.out.print(" ");

      ((D) o3).f();
      System.out.print(" ");

      ((C) o4).f();
      System.out.print(" ");

      ((C) o5).f();
    }
}
```

12. (3 b) Metóda garantuje, že po výpočte vráti celé číslo. Metóda, ktorá ju prekonáva, vracia len celé záporne čísla.

Týmto sa dôsledky a predpoklady pôvodnej metódy zoslabujú, zosilňujú alebo nemenia? Je týmto dodržaný Liskovej princíp substitúcie (LSP)? Je v tomto prípade vhodne použité dedenie?

Odpovedzte vo forme: *predpoklady / dôsledky / LSP / dedenie*. Položky *predpoklady* a *dôsledky* nahraďte jednou z možností *zoslabujú sa*, *zosilňujú sa* alebo *nemenia sa*. Položku *LSP* nahraďte jednou z možností *dodržaný* alebo *nedodržaný*. Položku *dedenie* nahraďte jednou z možností *áno* alebo *nie*.

(vyplňte tlačeným písmom)

Priezvisko:

Meno:

13. (10 b) Virtuálny priestor pozostáva z poprepájaných buniek. Bunka môže byť nečleniteľná alebo členiteľná. Členiteľná bunka môže obsahovať ďalšie nečleniteľné a členiteľné bunky. Bunky môžu byť prepojené bez ohľadu na to, či sú nečleniteľné alebo členiteľné. Pri každej bunke je možné získať zoznam buniek, do ktorých možno z tejto bunky vstúpiť, čo sú bunky, s ktorými je daná bunka priamo prepojená, a – v prípade členiteľnej bunky – bunky, z ktorých daná bunka bezprostredne pozostáva (prvá úroveň).

Základný návrh riešenia predložte vo forme náčrtu diagramu tried v UML, ktorý bude obsahovať najvýznamnejšie vzťahy, operácie a atribúty. Využite pritom najvhodnejší z návrhových vzorov Strategy, Observer, Visitor a Composite. Identifikujte explicitne prvky, ktorými sú modelované a implementované roly aplikovaného návrhového vzoru. Viditeľnosť prvkov nie je potrebné uvádzať.

Uveďte príslušnú implementáciu v Jave (samotný kód). V implementácii sa sústreďte na aplikačnú logiku – používateľské rozhranie nie je predmetom otázky. Zohľadnite princípy objektovo-orientovaného programovania. Poskytnite príklad použitia, v ktorom vytvoríte príslušné objekty a spustíte ich interakciu.

Odpoveď bude hodnotená podľa nasledujúceho kľúča:

- zabezpečenie základnej funkčnosti – 4 b
- kvalita a flexibilita objektovo-orientovaného návrhu – 6 b

Objektovo-orientované programovanie 2022/23
doc. Ing. Valentino Vranić, PhD., ÚISI FIIT STU
Skúška – opravný termín – 19. jún 2023
30

1 a

2 b

3 e

4 b

5 e

6 a

7 C.run(), M.b(), M.main()

8 b

9 c

10 a

11 F CDE F CDE CD (akceptovaná je aj odpoveď bez medzier)

12 nemenia sa / zosilňujú sa / neporušuje / áno

13 Mal byť aplikovaný vzor Composite. Bunka ako taká bola v role Component (rozhranie alebo abstraktná trieda), členiteľná bunka v role Composite, a nečleniteľná bunka v role Leaf (triedy odvodené od triedy reprezentujúcej bunku). Operácia výpisu alebo vrátení spojení mala byť implementovaná tak, aby sa dala uplatniť transparentne aj nad členiteľnou, aj nad nečleniteľnou bunkou (mala byť prítomná už v bunke, a v členiteľnej a nečleniteľnej bunke prekonaná).

Aj členiteľná, aj nečleniteľná bunka mali obsahovať evidenciu (napr. zoznam alebo pole) buniek, s ktorými sú spojené. Členiteľná bunka mala navyše obsahovať evidenciu buniek, z ktorých pozostáva. Operácia výpisu alebo vrátení spojení pri nečleniteľnej bunke mala vrátiť len bunky, s ktorými je daná bunka spojená, kým pri členiteľnej bunke k tomu mali pribudnúť aj bunky, z ktorých táto pozostáva.