TENTAMEN: Objektorienterad programmering

Läs detta!

- *Uppgifterna är inte ordnade efter svårighetsgrad.*
- Börja varje hel uppgift på ett nytt blad.
- Ordna bladen i uppgiftsordning.
- Skriv din tentamenskod på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Programkod skall skrivas i Java 5, eller senare version, och vara indenterad och renskriven.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor etc. får ej ändras.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

Välj **ett** alternativ för varje fråga! Garderingar ger noll poäng. Inga motiveringar krävs. Varje korrekt svar ger två poäng. Besvara delfrågorna 1.1-1.5 på ett blad.

1. Givet följande klasser

```
public  class C {
    public void f() { h(); }
    public void g() { System.out.println("C.g"); }
    public void h() { System.out.println("C.h"); }
}
public class B extends C {
    public void f(int x) { g(); }
    public void g() { System.out.println("B.g"); }
    public void h(int x) { System.out.println("B.h"); }
}
public class A extends B {
    public void f(char x) { System.out.println("A.f"); }
    public void g() { System.out.println("A.g"); }
    public void h() { System.out.println("A.h"); }
}
```

Vad skrivs ut om följande exekveras?

```
B a = new A();
a.f(42);
C b = new B();
b.f();
```

a. Först `B.g`, sen `C.h`
b. Först `A.g`, sen `C.h`
c. Först `A.g`, sen `B.h`
d. Först `B.g`, sen `A.h`

2. Givet klasserna

```
public class Base
public class Sub extends Base
```

Vad kan hända om man exekverar följande?

```
Base ref1 = objekt av lämplig typ;
Sub  ref2 = (Sub)ref1;
```

a. Det blir inga komplikationer tack vare typomvandlingen.
b. Kompilatorn signalerar ett typfel.
c. Undantaget `NullPointerException` kan kastas.
d. Undantaget `ClassCastException` kan kastas.

3. I objektorienterad programutveckling försöker man uppnå
a. hög kohesion och hög koppling.
b. låg kohesion och låg koppling.
c. låg kohesion och hög koppling.
d. hög kohesion och låg koppling.

4. Vad skrivs ut av resp. kodavsnitt om `f` kastar `IOException (=IOE)` resp. `NullPointerException (=NPE)`?

handler1
```
try { f(); }
catch ( Exception e)   {
    System.out.print("err1");
}
catch ( IOException e) {
    System.out.print("err2");
}
```

handler2
```
try { f(); }
catch ( IOException e)   {
    System.out.print("err3");
}
catch ( Exception e) {
    System.out.print("err4");
}
```

   a. handler1: `err1` i båda fallen, handler2: `IOE -> err3`, `NPE -> err4`.
   b. handler1: `IOE -> err2`, `NPE -> err1`, handler2: `IOE -> err3`, `NPE -> err4`.
   c. handler1 är ej kompilerbar, handler2: `IOE -> err3`, `NPE -> err4`.
   d. handler1 är ej kompilerbar, handler2: `IOE -> err3`, men `NPE` skickas vidare.

5. Givet är två paket:

```
package module1;

public class C1 {
    private int x;
    int y;
    protected int z;
}

class C3 {
    ...
}
```

```
package module2;
import module1.*;

public class C2 extends C1 {
    ...
}
```

Vilket påstående om synligheten hos variablerna `x`, `y` resp. `z` i `C1` är korrekt?

   a. `y` och `z` är synliga i `C3`, men inte `x`. Bara `z` är synlig i `C2`.
   b. `y` är synlig i `C3`, men inte `x` eller `z`. Bara `z` är synlig i `C2`.
   c. Alla tre variablerna är synliga i `C3`. Ingen är synlig i `C2`.
   d. `y` och `z` är synliga i `C2` och `C3`, men inte `x`.
   e. `y` och `z` är synliga i `C3`, men inte `x`. Ingen är synlig i `C2`.

(10 p)

Uppgift 2

Antag att man vill göra olika beräkningar på datasamlingar med numeriska värden. Det kan t.ex. gälla elementsumman, max- eller minvärdet, aritmetiskt medelvärde etc. i listor eller mängder. Vi kan införa klassen `CollectionUtils` med diverse metoder för att lösa problem av ovanstående slag. I denna uppgift skall vi konstruera metoden `compute`:

```
public class CollectionUtils {

    public static double compute(Collection<Double> collection,
                                 Computer comp)
    throws UndefinedOperationException { ... }

}
```

Metoden skall med hjälp av objektet `comp` utföra en beräkning på alla elementen i `collection`. Det beräknade värdet skall returneras. Typen `Computer` definieras

```
public interface Computer {
    void addValue(double value);
    double getValue() throws UndefinedOperationException;
}
```

Ett objekt av en implementerande subklass utför någon form av beräkning på värden som den får via metoden `addValue`. Resultatet av beräkningen kan hämtas med `getValue`. Om värdet är odefinierat, t.ex. på grund av att inga värden getts med `addValue` kastas undantaget `UndefinedOperationException`. Ex:

```
public class Sum implements Computer
```

Ett objekt av klassen håller reda på summan av inrapporterade värden och `getValue` returnerar summan. Vi kan på analogt sätt definiera ytterligare sådana klasser. Ex (returvärdena till höger):

```
ArrayList<Double> l = new ArrayList<>();
l.add(1.25);l.add(2.5);l.add(0.5);
CollectionUtils.compute(l,new Sum());                  4.25
CollectionUtils.compute(l,new ArithmeticMean());       1.4166666 …
CollectionUtils.compute(l,new Max());                  2.5
```

a)
Skriv metoden `compute` i klassen `CollectionUtils`.

(7 p)

b)
Konstruera klassen `ArithmeticMean`. Ett objekt av klassen skall hålla reda på det aritmetiska medelvärdet av de hittills givna värdena. Undantag skall kastas vid behov enligt ovan. Klassen får ej använda några datasamlingar utan endast enkla variabler av numerisk typ.
Ex:

```
Computer c = new ArithmeticMean();
c.addValue(3);
c.addValue(1);
c.addValue(6);
c.addValue(2);
c.getValue();         3
```

(7 p)

Uppgift 3

Medlemskap i viss klubb representeras med objekt av följande typ:

```
public class Membership {
    private String name;
    private String email;  // Primary key.
    private int born;      // Year of birth.

    public Membership(String name,String email,int born) {
        this.name = name;
        this.email = email;
        this.born = born;
    }
    public String getName()  { return name; }
    public String getEmail() { return email; }
    public int    getBorn()  { return born; }
    public void addFriend(String id) { ... }
}
```

Medlemmar identifieras unikt med sin e-postadress. Bygg ut `Membership` så att en medlem kan ha vänner. Vännernas e-postadresser skall lagras i en mängd (ej lista).

a) Deklarera en instansvariabel av lämplig typ och initiera den.

(2 p)

b) Implementera metod `getFriends` som returnerar en iterator till mängden av vänner.

(1 p)

c) Implementera metoden

```
public boolean removeFriend(String id)
```

som tar bort angivet vän-id ur mängden. Returvärdet skall avspegla om något togs bort eller inte.

(4 p)

En klubb representeras av klassen `Club`. Lagra medlemmarna i en tabell som avbildar medlems-id (e-post) på medlemsobjekt.

d) Deklarera tabellen av lämplig typ och initiera den.

(2 p)

e) Implementera metoden

```
public boolean addMember(Membership m)
```

som adderar `m` till tabellen. Om `m` redan är medlem skall ingen förändring ske. Returvärdet skall avspegla om objektet lades in i tabellen.

(2 p)

f) Implementera metoden

```
public boolean connect(String id1,String id2)
```

Metoden skall, under förutsättning att både `id1` och `id2` är medlemmar, addera `id1` till `id2`:s vänner och vice versa. Returvärdet skall avspegla om tabellen förändrades.

(2 p)

g) Implementera metoden

```
public boolean removeMember(String id)
```

Metoden skall, under förutsättning att `id` är medlem, ta bort denna ur tabellen. Dessutom skall `id` tas bort ur `id`:s vänners vänlistor. Returvärdet skall avspegla om tabellen förändrades. (4 p)

Uppgift 4

Skapa klassen `RandomString` som subklass till `java.util.Random` så att man enkelt kan tillverka strängar med slumpmässigt innehåll. Klassen skall ha en konstruktor som tar en teckensträng som inparameter samt metoden

```
public String nextString(int length)
```

Som returvärde skall man få en sträng av längd `length` bestående av tecken som valts slumpmässigt bland dem som gavs till konstruktorn.
Ex:

```
RandomString r = new RandomString("ABCDE1234567890");
System.out.println(r.nextString(6));          utskrift:   BAC60B
```

Tips: Det kan vara praktiskt att kopiera teckensträngen som är inparameter till konstruktorn till ett teckenfält (se strängklassen).

(10 p)

Uppgift 5

I ett kundregister hos ett företag används följande klasser för att lagra information om kunderna:

```
public class Customer {
    private int customerId;
    private String name;
    private Contact contact;
    …
}
public class Contact {
    private String address;
    private String email;
    private String phone;
    …
}
```

Kunder identifieras med kundnummer (`customerId`) och vi betraktar två kundobjekt som lika om de innehåller samma kundnummer. Implementera följande metoder i `Customer`, och i förekommande fall `Contact`, enligt de principer som lärts ut i kursen:

a) `equals`. Metoden skall ej vara överskuggningsbar vid arv.

(4 p)

b) `hashCode`.

(1 p)

c) `clone`. Objektet skall kopieras djupt.

(4 p)