CHALMERS

Institutionen för data- och informationsteknik

TENTAMEN

KURSNAMN	**Objektorienterad programmering, 7.5p**
PROGRAM:	**TKIEK-2, TKTFY-3, TKTEM-3** **2018/2019, lp 2**
KURSBETECKNING	**TDA550**
EXAMINATOR	**Uno Holmer**
TID FÖR TENTAMEN	**Onsdagen den 24/4 2019, 08.30 – 12.30**
HJÄLPMEDEL	**Java API** (delas ut av skrivningsvakten)
ANSV LÄRARE	**Uno Holmer** **tel. 772 5730** **besöker tentamen ca kl. 9.30 samt ca 11.30**
DATUM FÖR ANSLAG	**Senast den 24/5 2019** **Datum för granskning meddelas på kursens hemsida**
ÖVRIG INFORM.	**Betygsgränser: 3 - 24p, 4 - 36p, 5 - 48p. (max 60p)**

TENTAMEN: Objektorienterad programutveckling, fk

Läs detta!

- *Uppgifterna är inte ordnade efter svårighetsgrad.*
- Börja varje hel uppgift på ett nytt blad. Skriv inte i tesen.
- 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!**
- Skriv inte med rödpenna.
- 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 inte ändras. Fråga i oklara fall!
- 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

Klassen `Rectangle` bryter mot viktiga designprinciper, nämn de två mest relevanta.

```
public class Rectangle {
    public int width;
    public int height;
    public int getArea() { return width*height; }
}
```

En klient som använder klassen kan t.ex. se ut så här:

```
public class Client {
    public static void main(String[] arg) {
        Rectangle r = new Rectangle();
        r.width = 7;
        r.height = 6;
        int a = r.getArea();
        setSize(r,r.width*2,r.height + r.width);
    }
    public static void setSize(Rectangle r,int w,int h) {
        r.width = w; r.height = h;
    }
}
```

Refaktorera båda klasserna!

(7 p)

Uppgift 2

Vi har hittat klassen `Point` i ett API och skrivit en egen polygonklass:

```
public class Point {
    private int x;
    private int y;
    public Point(int x,int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
}
```

```
public class Polygon {
    private ArrayList<Point> points;
    public Polygon() {
        points = new ArrayList<>();
    }
    public void addPoint(Point p) {
        points.add(p);
    }
    public Point get(int i) {
        return points.get(i);
    }
    // ...
}
```

`Point`-objekt kan uppenbarligen muteras men hur är det med `Polygon`? En klient bör inte kunna göra andra ändringar i en polygon än att lägga till nya punkter. Refaktorera `Polygon`**!** Du får också lägga till en lämplig metod till `Point`.

(5 p)

Uppgift 3

I en Svensk mjukvara för E-handel finns behov av att enkelt kunna få reda på vad kunden skall betala i svenska kronor, inklusive kostnader som frakt, tull och moms. Du har hittat några API:er hos transportörer, tull- och skatteverk m.m. som bland annat innehåller klasserna:

```
public class PostSyd {
    public static float porto(float weight) {
        return weight*100; // SEK
    }
    ...
}
```

```
public class EuroMail {
    public static float deliveryCost (float weight) {
        return weight*100; // EUR
    }
    ...
}
```

```
public class USMail {
    public static float deliveryCost (float weight) {
        return weight*30; // USD
    }
    ...
}
```

```
public class Tullverket {
    public static float tullsats(String land) {
        if ( land.equals("USA") ) return 6; // %
        ...
    }
    public static int avgift() { return 100;  // SEK }
    ...
}
```

```
public class Skatteverket {
    public static int momssats() { return 25;  // % }
    ...
}
```

```
public class FårX {
    // Currency can be USD, EUR, SEK, ...
    public static float convert(String fromCurrency,String toCurrency)
    ...
}
```

Tillämpa designmönstret *Façade* så att programmeraren får en för ändamålet mer användbar metoduppsättning än vad ovanstående klasser ger. Fasaden skall ha tre metoder för att räkna ut slutpriset inkl. frakt, skatt och avgifter för en försändelse, en för köp i Sverige, en för Euroområdet, samt en för USA. Metoderna skall ta varornas sammanlagda pris och vikt som parametrar.

Totalpriser beräknas (starkt förenklat) så här: För leveranser i Sverige antas priset vara ex moms. Momsen adderas till varuvärdet och portot adderas. För Europeiska leveranser är momsen inräknad i priset. Fraktkostnaden adderas till varuvärdet och beloppet omvandlas till svenska kronor. För leveranser från USA adderas först tullavgiften till varuvärdet, sedan frakten. Beloppet omvandlas till svenska kronor och Tullverkets administrationsavgift läggs till. Därefter läggs moms på hela beloppet. Amerikanska priser är i USD, Europeiska i EUR och svenska i SEK. Alla viktangivelser avser kilogram.

(6 p)

Uppgift 4

Medianen i en sorterad datasamling är det mittersta elementet om samlingen har ett udda antal element och medelvärdet av de två mittersta elementen om antalet är jämnt. Nedan finns tre implementationer som beräknar medianen i ett fält, samt tre klienter som kan använda implementationerna.

```
public interface Imedian {
    double computeMedian(double a[]);
}

public class Impl1 implements Imedian {
    // @Pre: a refers to a non-empty sorted array
    // @Return: the median of the elements in a
    public double computeMedian(double a[]) { ... }
}

public class Impl2 implements Imedian {
    // @Pre: a refers to a non-empty array
    // @Return: the median of the elements in a
    public double computeMedian(double a[]) { ... }
}

public class Impl3 implements Imedian {
    // @Pre: True
    // @Return: the median of the elements in a or
    //          -1 if a does not refer to a non-empty sorted array
    public double computeMedian(double a[]) { ... }
}
```

a) I vilken av klasserna ovan har metoden `computeMedian` starkast specifikation? Är några specifikationer ojämförbara med avseende på styrka? Motivera svaret!

(4 p)

b) Ange för varje kombination av i och j vilken typ av resultat anropet $\texttt{client}_i\texttt{(new Impl}_j\texttt{())}$ kan förväntas ge (gärna i tabellform). Välj bland resultattyperna:

A. *computeMedian returnerar en median*
B. *det är ospecifierat vad computeMedian returnerar*
C. *ett undantag kastas troligen i computeMedian*
D. *computeMedian returnerar -1*

```
public static void client1(Imedian ms) {
    double[] arr = {3,1,4,5,2,6};
    System.out.println(ms.computeMedian(arr));
}
public static void client2(Imedian ms) {
    double[] arr = {1,2,3,4,5,6};
    System.out.println(ms.computeMedian(arr));
}
public static void client3(Imedian ms) {
    double[] arr = {};
    System.out.println(ms.computeMedian(arr));
}
```

(9 p)

Uppgift 5

a)

Studera följande kod:

```
public abstract class A {
    private float y;
    public A(float z) { y = z; }
    public abstract float f(float x);
    public float g(float x) { return h(x - y); }
    public float h(float x) { return x+3; }
}

public class B extends A {
    public B(float x) { super(x); }
    public float f(float x) { return 100 - g(x); }
    public float h(int x) { return x*3; }
}

public class C extends A {
    public C(float x) { super(x); }
    public float f(float x) { return 100 + g(x); }
    public float h(float x) { return x*2; }
}

public class Main {
    public static void main(String[] args) {
        A[] arr = new A[]{new B(205),new C(205)};
        float result = 0;
        for ( A obj : arr )
            result = result + obj.f(50);
        System.out.println(result);
    }
}
```

Vad skriver `main` ut? Förklara också vilka metoder som anropas – och varför, med vilka parametervärden, samt vad anropen returnerar. För att ange vilken klass en metod hör till kan du skriva t.ex. A.g, B.f, C.h o.s.v..

(4 p)

Uppgift b på nästa sida

b) Vilka av metodanropen är tillåtna och vilka ger kompileringsfel? Motivera svaren!

```
Base obj1 = new Sub1();
obj1.f();
obj1.g();
obj1.h();
Base obj2 = new Sub2();
obj2.f();
obj2.g();
obj2.h();
Int obj3 = new Sub2();
obj3.f();
obj3.g();
obj3.h();
```

```
public interface Int {
     public void h();
     public void f();
}
```

```
public class Base {
     public void f() {}
     public void g() {}
}
```

```
public class Sub1 extends Base {
     public void h() {}
}
```

```
public class Sub2 extends Base implements Int {
     public void h() {}
}
```

(3 p)

Uppgift 6

I systemprogram som hanterar användaridentiteter och lösenord brukar man inte lagra lösenorden i klartext utan i krypterad form. Man beräknar en s.k. hashsträng som beror av lösenordet. Hashsträngen har egenskapen att det är i det närmaste omöjligt att räkna ut lösenordet från den. Försök att knäcka säkerheten bygger därför oftast på att gissa olika lösenord och se om de ger samma hashsträng som det krypterade lösnordet för en viss användare.

Exempel: En viss hashalgoritm ger med lösenordet `querty` som indata hashsträngen
`35029a33a4321a1577432430a8520a406d1de641bea7dd96ade6311a90050a81`

Ändras lösenordet till `quertz` fås istället hashsträngen
`0672460aa483d27d77ac7a6976ca6933cac23f0f24b2d8a7f566a361c70010b2`

Små ändringar i lösenordet ger alltså helt olika hashsträngar.

Användarnamn och krypterade lösenord kan lagras externt i en kolonseparerad textfil:

```
password.txt
Lisa:35029a33a4321a1577432430a8520a406d1de641bea7dd96ade6311a90050a81
Rune:0672460aa483d27d77ac7a6976ca6933cac23f0f24b2d8a7f566a361c70010b2
...
```

Internt i programmet lagras informationen lämpligen i en tabell (map). För att beräkna hashsträngar från lösenord kan du använda klassen:

```
public class HashUtil {
    public static String secureHash(String str)
}
```

Konstruera klassen:

```
public class PasswordManager {
    public ... getInstance()
    public void addNewUser(String userId,String password)
    public boolean checkPassword(String userId,String password)
    public void changePassword(String userId,String oldPassword,
                                             String newPassword)
    public void save(String fileName) throws IOException
}
```

Metoderna:

`addNewUser`	lägger in en ny användare, samt lösenordet i krypterad form, i tabellen. Ev. befintlig användare med samma namn ersätts av den nya.
`checkPassword`	kontrollerar om `userId` har lösenordet `password`.
`changePassword`	byter lösenord för `userId` till `newPassword` förutsatt att `oldPassword` är samma som det befintliga lösenordet.

När klassen instansieras skall det undersökas om det finns en lösenordsfil med namnet `password.txt`, i så fall läses den in och lagras i den interna tabellen. Klassen skall utformas enligt principerna ovan och designmönstret *Singleton*.

(8 p)

Uppgift 7

Gränssnittet `IScanner` motsvarar en kraftigt bantad version av `java.util.Scanner`:

```
public interface IScanner {
    boolean hasNextLine();
    String nextLine();
    void close();
}
```

Antag att du har implementerat denna klass:

```
public class MyScanner implements IScanner {
    public MyScanner(File f) throws FileNotFoundException { ... }
    ...
}
```

Tillämpa designmönstret *Decorator* och konstruera klassen `ToUpperCaseTranslator` som översätter alla små bokstäver i texten den läser till stora. Metoden `nextLineUC` skall fungera som `nextLine` med skillnaden att den översätter små bokstäver till stora. Man skall t.ex. kunna använda klasserna så här:

```
public static void main(String[] arg) throws IOException { // Osnyggt!
    ToUpperCaseTranslator toUC =
        new ToUpperCaseTranslator(
            new MyScanner(
                new File("IScanner.java")));
    while ( toUC.hasNextLine() ) {
        System.out.println(toUC.nextLineUC());
    }
    toUC.close();
}
```

När ovanstående program körs skall utskriften bli:

```
PUBLIC INTERFACE ISCANNER {
    BOOLEAN HASNEXTLINE();
    STRING NEXTLINE();
    VOID CLOSE();
}
```

Tips: Du får också skapa en abstrakt basklass om det är lämpligt.

Användbara metoder:

```
boolean Character.isLowerCase(char c)
char Character.toUpperCase(char c)
```

(8 p)

Uppgift 8

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

```
public class Customer {
    private String 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 likadana kundnummer. Implementera följande metoder i `Customer`, och om lämpligt även i `Contact`, enligt de principer som lärts ut i kursen:

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

(1 p)

b) `hashCode`.

(1 p)

c) `clone`. Objektet skall kopieras djupt.

(2 p)

d) `toString`. Utskriften skall utformas i stil med:

```
Lisa Nilsson (Customer: 12345)
Contact:
Address: Lisastigen 2
Email:   nilsson@nmail.se
Phone:   0123-77788899

Olle Olsson (Customer: 67890)
Contact:
Address: Olsängen 34
Email:   olleo@omail.se
Phone:   0451-1127839
```

(2 p)

Lösningsförslag till tentamen

Kurs	**Objektorienterad programutveckling, fk**
Tentamensdatum	**2019-04-24**
Program	**TKIEK-2,TKTFY-3,TKTEM-3**
Läsår	**2018/2019, lp 2**
Examinator	**Uno Holmer**

Uppgift 1 (7 p)

Klassen `Rectangle` bryter mot *Uniform access principle*. Klienter får läsa av bredd och höjd via två instansvariabler men för arean finns en metod – inkonsekvent. Dessutom bryter den mot *Information hiding principle* eftersom instansvariablerna är publika.
(Frågan avser inte klienten men som helhet betraktat bryter designen också mot *Information expert principle* och *Tell don't ask*, `setSize` borde naturligtvis tillhöra `Rectangle`.)

Refaktorering ger förslagsvis följande:

```
public class Rectangle {
    private int width;
    private int height;
    public Rectangle(int width,int height) {
        this.width = width;
        this.height = height;
    }
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public int getArea() {
        return width*height;
    }
    public void setSize(int width,int height) {
        this.width = width;
        this.height = height;
    }
}
public class Client {
    public static void main(String[] arg) {
        Rectangle r = new Rectangle(7,6);
        int a = r.getArea();
        r.setSize(r.getWidth()*2,r.getHeight() + r.getWidth());
    }
}
```

Uppgift 2 (5 p)

Klassen `Polygon` är muterbar eftersom klienten som skapar och lägger till punkter eller hämtar punkter kan modifiera dessa via sparade referenser (alias). Punkter som adderas till polygonen bör ägas exklusivt av denna. För att undvika aliasproblemet låter vi konstruktorn och accessmetoden klona de berörda punktobjekten.

Lägg först till en `clone`-metod till `Point`:

```
public class Point implements Cloneable
...
public Point clone() {
    try {
        return (Point)super.clone();
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}
```

och modifiera konstruktorn och `get`-metoden i `Polygon`:

```
public class Polygon {
    private ArrayList<Point> points;
    public Polygon() {
        points = new ArrayList<>();
    }
    public void addPoint(Point p) {
        points.add(p.clone());
    }
    public Point2 get(int i) {
        return points.get(i).clone();
    }
}
```

Uppgift 3 (6 p)

```
public class PriceHunter {
    public static float calculatePriceSE(int priceSEK,int weight) {
        return priceSEK*(1 + Skatteverket.momssats()/100.0F) +
                PostSyd.porto(weight);
    }

    public static float calculatePriceEuro(int priceEUR,int weight) {
        return (priceEUR + EuroMail.deliveryCost(weight))*
            ExchangeRates.convert("EUR","SEK");
    }

    public static float calculatePriceUS(int priceUSD,int weight) {
        return ((priceUSD*(1 + Tullverket.tullsats("USA")/100.0F) +
                    USmail.deliveryCost(weight)) *
                        ExchangeRates.convert("USD","SEK") +
                Tullverket.avgift()) *
                (1 + Skatteverket.momssats()/100.0F);

    }
}
```

Uppgift 4 (4+9 p)

a)

Metoden `computeMedian` har starkare specifikationen i `Impl2` än i `Impl1` eftersom den har ett svagare förvillkor. Specifikationen i `Impl3` är ojämförbar med de andra eftersom både dess förvillkor och eftervillkor är svagare än deras.

b)

	Impl1	Impl2	Impl3
client1	B	A	D
client2	A	A	A
client3	B	B	D

Uppgift 5 (4+3 p)

a)
Utskriften blir 42.0. Det första fältelementet är ett `B`-objekt och det andra ett `C`-objekt. Loopvariabelns dynamiska typ går alltså från `B` till `C`. Klasserna `B` och `C` överskuggar metoden `f` och `C` dessutom h varför de metoderna väljs om de anropas för objekt av resp. typ (dynamisk bindning). Observera att `B.h(int x)` inte överskuggar `A.h(float x)`. Metoden `g` ärvs alltid från `A`. Instansvariabeln `A.y` initieras i båda fallen till 205.

Första loopvarvet:

```
B.f(50.0)  →    A.g(50.0)  →    A.h(-155.0)
252.0      ←    -152.0     ←    -152.0
```

Andra loopvarvet:

```
C.f(50.0)  →    A.g(50.0)  →    C.h(-155.0)
-210.0     ←    -310.0     ←    -310.0
```

b)

`Base obj1 ...`	objektet som `obj1` refererar till är irrelevant för statisk typcheckning
`obj1.f();`	OK
`obj1.g();`	OK
`obj1.h();`	Otillåtet, den statiska typen `Base` saknar `h`
`Base obj2 ...`	
`obj2.f();`	Samma
`obj2.g();`	som
`obj2.h();`	ovan
`Int obj3 ...`	
`obj3.f();`	OK
`obj3.g();`	Otillåtet, den statiska typen `Int` saknar `g`
`obj3.h();`	OK

Uppgift 6 (8 p)

```
public class PasswordManager {
    private Map<String,String> passwordMap;
    private static PasswordManager instance = null;
    private PasswordManager() {
        passwordMap = new TreeMap<>();
        try {
            load("password.txt");
        }
        catch ( IOException e ) {
            e.printStackTrace();
            System.exit(0);
        }
    }
    public static PasswordManager getInstance() {
        if ( instance == null )
            instance = new PasswordManager();
        return instance;
    }
    public void addNewUser(String userId,String password) {
        if ( ! passwordMap.containsKey(userId) ) {
            passwordMap.put(userId,HashUtil.secureHash(password));
        }
    }
    public boolean checkPassword(String userId,String password) {
        String hashedPwd = passwordMap.get(userId);
        if ( hashedPwd == null )
            return false;
        else
            return HashUtil.secureHash(password).equals(hashedPwd);
    }
    public void changePassword(String userId,String oldPassword,
                                             String newPassword)
    {
        if ( checkPassword(userId,oldPassword) )
            passwordMap.put(userId,HashUtil.secureHash(newPassword));
    }
    public void save(String fileName) throws IOException {
        PrintWriter pw = new PrintWriter(new FileWriter(fileName));
        for ( Map.Entry<String,String> e : passwordMap.entrySet() ) {
            pw.println(e.getKey() + ":" + e.getValue());
        }
        pw.close();
    }
    private void load(String fileName) throws IOException {
        File file = new File(fileName);
        if ( file.exists() )
            readFileToMap(file);
    }
    private void readFileToMap(File file) throws IOException {
        Scanner sc = new Scanner(file);
        while ( sc.hasNextLine() ) {
            String[] fields = sc.nextLine().split(":");
            if ( fields.length != 2 )
                throw new IOException("Corrupt password file");
            passwordMap.put(fields[0],fields[1]);
        }
        sc.close();
    }
}
```

Uppgift 7 (8 p)

```
public abstract class AbstractScannerDecorator implements IScanner {
    private IScanner decoratedScanner;
    public AbstractScannerDecorator(IScanner decoratedScanner) {
        this.decoratedScanner = decoratedScanner;
    }
    @Override
    public boolean hasNextLine() {
        return decoratedScanner.hasNextLine();
    }
    @Override
    public String nextLine() {
        return decoratedScanner.nextLine();
    }
    @Override
    public void close() {
        decoratedScanner.close();
    }
}

public class ToUpperCaseTranslator extends AbstractScannerDecorator {
    public ToUpperCaseTranslator(IScanner decoratedScanner) {
        super(decoratedScanner);
    }
    public String nextLineUC() {
        String src = nextLine();
        StringBuilder sb = new StringBuilder();
        for ( int i = 0; i < src.length(); i++ ) {
            char c = src.charAt(i);
            if ( Character.isLowerCase(c) )
                sb.append(Character.toUpperCase(c));
            else
                sb.append(c);
        }
        return sb.toString();
    }
```

Den observante inser förstås att detta kan göras mycket enklare med strängklassens egen toUpperCase-metod: ☺

```
    public String nextLineUC() {
        return nextLine().toUpperCase();
    }
}
```

Uppgift 8 (1+1+2+2 p)

a) Vi `final`-deklarerar `equals` för att förhindra överskuggning vid arv:

```
public final boolean equals(Object other) {
    if ( this == other )
        return true;
    else if ( other instanceof Customer ) {
        return ((Customer)other).customerId.equals(customerId);
    } else
        return false;
}
```

b) I `hashCode` delegerar vi helt enkelt vidare till strängklassens motsvarighet:

```
public int hashCode() {
    return customerId.hashCode();
}
```

c) Metoden `clone` krävs i både `Contact` och `Customer` och båda skall implementera gränssnittet `Cloneable`.

```
public Customer clone() {
    try {
        Customer copy = (Customer)super.clone();
        copy.contact = contact.clone();
        return copy;
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}
public Contact clone() {
    try {
        return (Contact)super.clone();
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}
```

d)

```
public String toString() {
    return name + " (Customer:  " + customerId + ")\n" +
           "Contact: \n" + contact;
}
```