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;
}
```