Tentamen för TDA550 Objektorienterad programvaruutveckling IT, fk

DAG: 14-04-22 TID: 8:30 – 12:30

Ansvarig: Christer Carlsson, ankn 1038

Förfrågningar: Christer Carlsson

Resultat: erhålls via Ladok

Betygsgränser:

3:a	24 poäng
4:a	36 poäng
5:a	48 poäng
maxpoäng	60 poäng

Siffror inom parentes: anger maximal poäng på uppgiften.

Granskning: Onsdag 14/5 kl 12-13 och fredag 16/5 kl 12-13, rum 6128 i EDIT-huset.

Hjälpmedel: Inga hjälpmedel är tillåtna förutom bilagan till tesen.

Var vänlig och: Skriv tydligt och disponera papperet på lämpligt sätt.

Börja varje uppgift på nytt blad. Skriv ej på baksidan av papperet.

Observera: Uppgifterna är ej ordnade efter svårighetsgrad. Titta därför igenom hela tentamen innan du börjar skriva.

Alla program skall vara välstruktruerade, lätta att överskåda samt enkla att förstå.

Vid rättning av uppgifter där programkod ingår bedöms principella fel allvarligare än smärre språkfel.

LYCKA TILL!!!!

Uppgift 1.

Betrakta klasserna Bird och Penguin som finns i paketet aviator:

```
package aviator;
public abstract class Bird {
    protected int a = 1;
    public abstract void identify();

    protected void shout(){
        System.out.println("Kaw-Kaw");
    }

    public void action() {
        System.out.println("Fear me" + this.a + " times");

    }

    public void flyAround() {
        System.out.println("Wooooosh, I am a");
        identify();
        shout();
        System.out.println("You should");
        action();
    }

    protected int getA() {
        return a;
    }

    protected void addToA() {
        a++;
    }
}//Bird
```

```
package aviator;
public class Penguin extends Bird {
    private int a;

    public Penguin() {
        a = 2;
    }

    @Override
    public void identify() {
        System.out.println("Penguin #" + a +
                           " reporting for duty!");
    }

    public void shout() {
        System.out.println("Shouting is uncivilized...");
    }

    public void action(int times) {
        System.out.println("Swim "+  (times + getA()) +
                           " times!");
    }

    public void flyAround() {
        System.out.println("Sometimes I dream" +
                           " I can fly like this: ");
        super.flyAround();
    }
} //Penguin
```

I paketet bungler finns klassen Program:

```
package bungler;
public class Program{
    public static void act1() {
        Bird b = new Bird();
        b.flyAround();
    }

    public static void act2() {
        Bird b = new Penguin();
        act3(b);
    }

    public static void act3(Penguin p) {
        Bird b  = new Penguin();
        b.shout();
    }

    public static void act4() {
        Bird b  = new Penguin();
        b.action(15);
    }

    public static void act5() {
        Bird b = new Penguin();
        b.flyAround();
    }

    public static void act6() {
        Bird b  = new Penguin();
        b.identify();
    }

    public static void act7() {
        Bird b = new Penguin();
        b.addToA();
        ((Penguin)b).action(4);
    }
}//Program
```

Vad inträffar för vart och ett av nedanstående anrop (ger kompileringsfel, ger exekveringsfel, ger utskriften ”xxxx”, etc). Svaren skall motiveras!

a) Program.act1() b) Program.act2() c) Program.act3(**new** Penguin()) d) Program.act4()

e) Program.act5(5) f) Program.act6() g) Program.act7()

Anm: Klassen Program går naturligtvis inte att kompilera, men bortse från detta när du besvarar frågorna. Tänk dej att både kompilering och exekvering görs när respektive metod anropas.

(7 poäng)

Uppgift 2.

Antag att vi har en klass BankAccount för att avbilda bankkonton:

```
public class BankAccount {
    private int balance;    // current account balance
    public void withdraw(int amount) {
        . . .
    }
    . . .
}//BankAccount
```

Betrakta följande specifikationer för metoden withdraw:

i) @effects decreases balance by amount

ii) @requires amount <= balance and amount >= 0
@effects decreases balance by amount

iii) @throws InsufficientFundsException if balance < amount
@effects decreases balance by amount

a) Vilka av ovanstående specifikationer uppfylls av implementationen nedan?

```
public void withdraw(int amount) {
    balance = balance - amount;
}
```

b) Vilka av ovanstående specifikationer uppfylls av implementationen nedan?

```
public void withdraw(int amount) {
    if (balance >= amount)
        balance = balance - amount;
}
```

c) Vilka av ovanstående specifikationer uppfylls av implementationen nedan?

```
public void withdraw(int amount) {
    if (amount < 0)
        throw new IllegalArgumentException();
    balance = balance - amount;
}
```

d) Vilka av ovanstående specifikationer uppfylls av implementationen nedan?

```
public void withdraw(int amount) throws InsufficientFundsException {
    if (amount < 0)
        throw new InsufficientFundsException();
    balance = balance - amount;
}
```

Ge en kort motivering till dina svar!

(6 poäng)

Uppgift 3.

Betrakta nedanstående klass:

```
public class Doer {
    private int iD;
    public Doer(int i) {
        iD = i;
    }

    public void doSomething() {
        switch (iD) {
            case 1: nothing();
                break;
            case 2: talk();
                break;
            case 3: laugh();
            default: return;
        }
    }

    private void laugh() {
        System.out.println("Laugh");
    }

    private void talk() {
        System.out.println("Talk");
    }

    private void nothing(){
        System.out.println("Nothing");
    }
}//Doer
```

Klassen strider mot *Open-Closed Principle*. Åtgärda detta genom att nyttja designmönstret Strategy.

(6 poäng)

Uppgift 4.

Betrakta nedanstående kod:

```
public interface EuroThermo {
    public double readTemp();
}//EuroThermo

public class USThermo {
    private double usTemp;
    public USThermo(double usTemp) {
        this.usTemp = usTemp;
    }
    public double getTemp() {
        return usTemp;
    }
}//USThermo
```

Interfacet EuroThermo definierar en termometer som mäter i Celsius och klassen USThermo implementerar en termometer som mäter i Fahrenheit. Utnyttja designmönstret Adapter för att implementera en klass USThermoToEuroThermo som anpassar en termometer som mäter i Fahrenheit till en termometer som mäter i Celsius. Skriva också en main-metod som testar din implementation.

Tips: Formeln för att omvandla från Fahrenheit (F) till Celsius (C) ser ut på följande sätt: C = (F-32)*(5/9).

(4 poäng)

Uppgift 5.

Betrakta klass Appointment nedan:

```
public class Appointment {
    private String description;
    private Date time;
    . . .
}
```

Utöka klassen med en clone-metod, samt ange vad som i övrigt behöver göras för att klassen skall bli kloningsbar.
Tips: Klassen Date tillhandahåller metoden clone.

(3 poäng)

Uppgift 6.

Kalle Klant har fått till uppgift att skriva en equals-metod till klassen StockItem nedan:

```
public class StockItem {
    private String name;
    private int size;
    private String description;
    private int quantity;
    public StockItem(String name, int size, String description, int quantity) {
        //code not shown here
    }

    /*Other constructors and methodes not shown here */
}
```

Två objekt av StockItem skall betraktas som lika om värden på instansvariablerna name respektive size överensstämmer. Kalle presenterade följande lösning:

```
/** return true if the name and size fields match */
public boolean equals(StockItem other) {
    return name.equals(other.name) && size == other.size;
}
```

a) Denna equals-metod ger inte alltid rätt resultat. Ge ett exempel på ett scenario där ett felaktigt resultat erhålls. (1 poäng)

b) Ge en korrekt implementation av equals-metoden. (3 poäng)

c) Vilka av nedanstående implementationer av hashCode() är korrekta? Motivera ditt svar!

i)
```
public int hashCode() {
    return name.hashCode();
}
```

ii)
```
public int hashCode() {
    return name.hashCode()*17 + size*11;
}
```

iii)
```
public int hashCode() {
    return name.hashCode()*17 + size*11+ description.hashCode()*7;
}
```

iv)
```
public int hashCode() {
        return name.hashCode()*17 + size*11+ description.hashCode()*7 + quantity*5;
}
```
(2 poäng)

d) Vilken av de korrekta implementationerna av hashCode() är att föredra? Motivera ditt svar! (1 poäng)

Uppgift 7.

I ett programsystem för att handha en djurpark finns nedanstående metod:

```
public void feedAnimals() {
    for (Animal a: animalsInPark) {
        if (a.needsFood()) {
            Food typeOfFood = a.typeOfFoodIfFoodNeeded();
            a.feed(typeOfFood);
        }
    }
}
```

Gör en refaktorering (omstrukturering) av koden så att principen *Separation of Concerns* följs. (4 poäng)

Uppgift 8.

En grupp av ord är *anagram* om de kan bildas av varandra genom att ändra ordningen på de ingående bokstäverna. Exempelvis är de svenska orden "avig" och "viga" anagram till varandra.

I denna uppgift skall delar av en klass för hantering av anagram implementeras:

```
public class Anagrams {
    private Map<String, Set<String>> anagrams;
    ...
    private String alphabetize(String word) {
        //code not shown here
    }
}
```

För att hålla reda på anagram används i klassen en Map. Värdena i mappen utgörs av mängder (Set) innehållande ord som är anagram med varandra, och nycklar i mappen är de strängar man får om man sorterar bokstäverna i anagramen i tillhörande Set i alfabetisk ordning. Om vi t.ex. skall sätta in orden "avig", och "viga" i en (från början tom) map görs följande:

- När ordet "avig" skall sättas in bildar vi först sträng som utgör nyckel genom att sortera bokstäverna i "avig". Vi får då nyckeln "agiv". Denna nyckel finns inte i mappen (som är tom). Nyckeln sätts då in i mappen med sitt tillhörande värde, mängden ["avig"].

- När "viga" skall sättas in bildar vi först sträng som utgör nyckel, vilken blir "agiv". Denna nyckel finns redan i mappen. Ordet "viga" läggs därför in i mängden som tillhör nyckel "agiv", som då för värdet ["avig", "viga"].

a) Lägg till och implementera en konstruktor i klassen. Den konkreta klassen HashMap skall användas. (1 poäng)

b) Implementera metoden

```
public void add(String word)
```

för att sätta in ordet word i mappen. Du kan använda en färdigskriven metod, String alphabetize(String word) som returnerar en sträng som består av samma bokstäver som i parametern, men i alfabetisk ordning. (3 poäng)

c) Implementera metoden

```
public Set<String> getAnagramsOf(String word)
```

för att ta reda på alla ord i mappen som är anagram till ordet word. Om inga anagram till word finns skall mängden som returneras vara tom, annars skall mängden som returneras bestående av alla ord som är anagram till word utom word självt.

Exempel 1: Antag att vi satt in orden "avig" och "viga" och anropar getAnagramsOf("giva").
Då skall resultatet bli en mängd ["avig", "viga"].

Exempel 2: Antag att vi satt in "eldig", "ledig" och "digel" och anropar getAnagramsOf("ledig").
Då skall resultatet bli en mängd ["digel", "eldig"]. (3 poäng)

d) Implementera metoden

```
public int maxGroupSize();
```

som tar reda på storleken av den största mängden anagram. Om mappen är tom skall 0 returneras. (2 poäng)

Uppgift 9.

Betrakta medanstående klass:

```
public class TryCatchMystery {

    public static void main (String[] args) {
        try {
            method1();
            method2();
        }
        catch (IllegalArgumentException e) {
            System.out.println("main IllegalArgumentException");
        }
        catch (RuntimeException e) {
            System.out.println("main RuntimeException");
        }
    }//main

    public static void method1() {
        System.out.println("entered method1");
        try {
            method2();k
        }
        catch (IllegalArgumentException e) {
            System.out.println("method1 IllegalArgumentException");
            throw new NullPointerException();
        }
        catch (NullPointerException e) {
            System.out.println("method1 NullPointerException");
            throw new NullPointerException();
        }
        finally {
            System.out.println("method1: finally");
        }
        System.out.println("exited method1");
    }//method1

    public static void method2() {
        System.out.println("entered method2");
        throw new IllegalArgumentException();
    }//method2
}//TryCatchMystery
```

Vad blir utskriften när main-metoden körs? Tips: IllegalArgumentException och NullPointerException är subklasser till RuntimeException. (2 poäng)

Uppgift 10.

I ett Java-program som används för att testa stresståligheten hos en grupp försökspersoner, går ett experimentet ut på att försökspersonen skall göra en förutbestämd följd av knapptryckningar. Om en knapptryckning blir felaktig startas en tråd av klassen ShutdownThread (se nedan) som skriver ut ett felmeddelande, varefter tråden lägger sig och sover en viss tid, under vilken försökspersonen får chansen att korrigera sitt fel (genom att ge en ny sekvens av knapptryckningar). Lyckas försökspersonen med detta inom tiden tråden sover, avbryts tråden med ett anrop av interrupt(), annars skjuter tråden ner programmen genom att anropa System.exit(0).

Din uppgift är att skriva klassen ShutdownThread. Klassen ShutdownThread skall ha har en konstruktor

```
public ShutdownThread(String msg, int seconds)
```

där parametern msg anger felmeddelandet som skrivs ut och parametern seconds anger hur länge försökspersonen har på sig att rätta till sitt fel innan systemet skjuts ner.

Tips: Om tråden blir avbruten görs lämpligen **return**, för att avsluta tråden på ett kontrollerat sätt. (8 poäng)

Uppgift 11.

Antag att vi har följande klasser:

```
public class Animal extends Object { ... }
public class Pet extends Animal { ... }
public class Cat extends Pet { ... }
public class Dog extends Pet { ... }
```

Antag vidare att man i ett program gjort följande deklarationer:

```
Animal a;
Pet p;
Cat c;
Dog d;
List <? extends Pet> lep;
List <? super Pet> lsp;
```

Ange för var och en av satserna nedan om satsen är korrekt eller ger typfel. Motivera ditt svar!

a) lep.add(p);

b) a = lep.get(0);

c) lsp.add(c);

d) a = lsp.get(0); (4 poäng)

Tentamen 140422 - LÖSNINGSFÖRSLAG

Uppgift 1.

a) Kompileringsfel! En abstrakt klass kan inte instaisieras.

b) Kompileringsfel! Metoden act3(Penguin p) kan inte anropas med en aktuell parameter av typen Bird.

c) Kompileringsfel! Metoden shout() är protected och anropas utanför paketet.

d) Kompileringsfel! Metoden action(**int** i) finns inte för den statiska typen Bird.

e) Kompileringsfel! Metoden act5 har ingen parameter.

Eftersom man från frågeställningen kunde få uppfattningen om att eventuella fel låg i klassen Program har jag även godkänt att anropet av act5 var felaktigt. Utskriften blir då:

```
Some times I dream I can fly like this:
Wooooosh, I am
Penguin #2 reporting for duty!
Shouting is uncivilzed...
You should
Fear me 1 times
```

f) Penguin #2 reporting for duty!

g) Kompileringsfel! Metoden addToA() är **protected** och anropas utanför paketet.

Uppgift 2.

a)

i) Ja! Metoden gör exakt vad specifikationen säger

ii) Ja! Om klienten följer förvillkoret (@requires) gör koden det som förväntas

iii) Nej! Metoden kastar inget exception.

b)

i) Nej! Balansen på kontor kommer inte alltid att minska

ii) Ja! Om klienten följer förvillkoret (@requires) gör koden det som förväntas

iii) Nej! Metoden kastar inget exception.

c)

i) Nej! Balansen på kontor kommer inte alltid att minska

ii) Ja! Om klienten följer förvillkoret (@requires) gör koden det som förväntas

iii) Nej! Metoden kastar fel typ av exception och av felaktig orsak.

d)

i) Nej! Balansen på kontor kommer inte alltid att minska

ii) Ja! Om klienten följer förvillkoret (@requires) gör koden det som förväntas

iii) Nej! Metoden kastar exception av felaktig orsak.

Uppgift 3.

```
public class Doer {
    private Strategy doing;
    public Doer(Strategy doing) {
        this.doing = doing;
    }
    public void doSomething() {
        doing.doSomething();
    }
}

public interface Strategy {
    public void doSomething();
}

public class Laugh implements Strategy {
    @Override
    public void doSomething() {
        System.out.println("Laugh");
    }
}

public class Talk implements Strategy {
    @Override
    public void doSomething() {
        System.out.println("Talk");
    }
}

public class Nothing implements Strategy {
    @Override
    public void doSomething() {
        System.out.println("Nothing");
    }
}//Doer
```

Uppgift 4.

```
public class USThermoToEuroThermo implements EuroThermo {
    private final USThermo ust;
    public USThermoToEuroThermo(USThermo ust) {
        this.ust = ust;
    }
    public double readTemp() {
        return (ust.getTemp() - 32.0) * (5.0 / 9.0);
    }
}
public class Test {
    public static void main (String[]args) {
        USThermo ust = new USThermo(150);
        USThermoToEuroThermo useu = new USThermoToEuroThermo(ust);
        System.out.println(useu.readTemp());
    }
}
```

Uppgift 5.

```
public class Appointment implements Cloneable {

    // som tidigare

    public Appointment clone() {
        try {
            Appointment copy = (Appointment) super.clone();
            copy.time = (Date) time.clone();
            return copy;
        }
        catch (CloneNotSupportedException e) {
            throw new InternalError();
        }
    }//clone
}//Appointment
```

Uppgift 6.

a)

Kalles lösning använder inte överskuggning utan överlagring. Nedanstående satser resulterar därför i att equals-metoden i klassen Object anropas:

```
Object s1 = new StockItem("thing", 1, "stuff", 1);
Object s2 = new StockItem("thing", 1, "stuff", 1);
System.out.println(s1.equals(s2));
```

Eftersom equals-metoden Object betecknar två objekt som lika endast om de alias fås inte det förväntade resultatet.

b)

```
/** return true if the name and size fields match */
@ Override
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null)
        return false;
    if (this.getClass() != o.getClass())
        return false;
    StockItem other = (StockItem) o;
    return name.equals(other.name) && size == other.size;
}
```

c)

i) och ii) är giltiga. iii) och iv) är ogiltiga eftersom equals-metoden inte använder instansvariabelerna description och quantity.

d) Alternativ ii) är att föredra då denna implementation ger en bättre spridning av hashkoderna, eftersom båda instansvariablerna size och name används vid beräkningen. Alternativ i) använder endast instansvariabeln name vid beräkningen av hashkoden, varför två objekt med samma namn får samma hashkod även om instansvariabeln size har olika värden.

Uppgift 7.

```
public void feedAnimals() {
    for (Animal a: animalsInPark) {
        feedIfNeeded(a);
    }
}

public void feedIfNeeded(Animal a) {
    if (a.needsFood()) {
        decideTypeAndFeed(a);
    }
}

private void decideTypeAndFeed(Animal a) {
    Food typeOfFood = a.typeOfFoodNeeded();
    a.feed(typeOfFood);
}
```

Uppgift 8.

a)

```
public Anagrams() {
    anagrams = new HashMap<String, Set<String>>();
}
```

b)

```
public void add(String word) {
    String key = alphabetize(word);
    Set<String> anagramSet = anagrams.get(key);
    if (anagramSet == null) {
        anagramSet = new TreeSet<String>();
        anagrams.put(key, anagramSet);
    }
    anagramSet.add(word);
}
```

c)

```
public Set<String> getAnagramsOf(String word) {
    Set<String> s = anagrams.get(alphabetize(word));
    if (s != null) {
        s = new TreeSet<String>(s);          // make a copy
        s.remove(word);                      // remove word, if present
        return s;
    }
    return new TreeSet<String>();
}
```

d)

```
public int maxGroupSize() {
    int largest = 0;
    for (Set<String> s : anagrams.values()) {
        if (s.size() > largest) {
            largest = s.size();
        }
    }
    return largest;
}
```

Uppgift 9.

Utskriften blir:

```
entered method1
entered method2
method1 IllegalArgumentException
method1: finally
main RuntimeException
```

Uppgift 10.

```
public class ShutdownThread extends Thread {
    private int secs;
    private String msg;

    public ShutdownThread(String msg, int secs) {
        this.secs = secs;
        this.msg = msg;
    }//constructor

    public void run() {
        System.out.println(msg);
        try {
            Thread.sleep(secs*1000);
        } catch (InterruptedException e) {
            return;
        }
        shutdownNow();
    }//run

    public static void shutdownNow() {
        System.out.println("You failed!");
        System.exit(0); // cause entire JVM to shut down
    }// shutdownNow
}// ShutdownThread
```

Om man väljer att implementera Runnable istället för att utöka Thread måste klassen tillhandahålla metoderna start() och interrupt():

```
public class ShutdownThread implements Runnable {
    private int secs;
    private String msg;
    private Thread thread = new Thread(this)
    public ShutdownThread(String msg, int secs) {
        this.secs = secs;
        this.msg = msg;
    }//constructor

    public void start() {
        thread.start();
    }

    public void interrupt() {
        thread.interrupt();
    }

    public void run() {
        System.out.println(msg);
        try {
            Thread.sleep(secs*1000);
        } catch (InterruptedException e) {
            return;
        }
        shutdownNow();
    }//run

    public static void shutdownNow() {
        System.out.println("You failed!");
        System.exit(0); // cause entire JVM to shut down
    }// shutdownNow
}// ShutdownThread
```

Uppgift 11.

a) Ger typfel. lep kan referera till objekt av följande typer: List<Pet>, List<Cat> och List<Dog>. Det går inte att lägga in ett objekt av typen Pet i en lista av typen List<Cat> eftersom Pet inte är en subtyp till Cat, och det går inte att lägga in ett objekt av typen Pet i en lista av typen List<Dog> eftersom Pet inte är en subtyp till Dog.

b) Korrekt. lep kan referera till objekt av följande typer: List<Pet>, List<Cat> och List<Dog>.. Det objekt som fås med lep.get(0) är alltså av typen Pet, Cat eller Dog. Variabeln a, som är av typen Animal, kan referera till samtliga dessa typer, eftersom de är subtyper till Animal.

c) Korrekt. lsp kan referera till följande typer: List<Pet>, List<Animal> och List<Object>. Eftersom Cat är subtyp till Pet, subtyp till Animal och subtyp till Objekt kan ett objekt av typen Cat läggas i samtliga dessa typer av listor.

d) Ger typfel. lsp kan referera till följande typer: List<Pet>, List<Animal> och List<Object>. Det objekt som fås med lsp.get(0) är alltså av typen Pet, Animal eller Object. Variabeln a, som är av typen Animal, kan inte referera till ett objekt av typen Object, eftersom Object inte är en subtyp till Animal.