Tentamen för TDA550 Objektorienterad programvaruutveckling IT, fk

DAG: 15-04-13 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: Måndag 11/5 kl 14-15 och tisdag 12/5 kl 14-15, 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 nedanstående klasser och interface:

```
public interface A {
   public void print();
}//A


public class B implements A {
   public void print() {
      System.out.println("In class B print");
   }

    public void fax() {
      System.out.println("In class B fax");
   }
}//B

 public class C implements A {
   public void print() {
      System.out.println("In class C");
   }

   public void fax() {
      System.out.println("In class C fax");
   }

   public void print(Object o) {
      System.out.println("In the object");
   }
}//C
```

```
public class D extends C {
   public double storage1;
   private double storage2;

   public D() {
      storage1 = 0;
      storage2 = 0;
   }

   public D(double x, double y){
      storage1 = x;
      storage2 = x;
   }

   public void print() {

      System.out.println("First print in D");
   }

   public static void print(double storage1, double storage2) {
      System.out.println("Second print in D");
    }

    private void print(int x) {
      System.out.println("Third print in D");
   }

   public void print(double y) {
      System.out.println("Fourth print in D");
   }

   public double getStorage1() {
      return storage1;
   }

   public double getStorage2() {
      return storage2;
   }
}//D
```

Vad blir resultatet för var och en av följande satser (ger kompileringsfel, ger exekveringsfel, skriver ut xxx, ect)? Motivera dina svar!

i)
```
D foo = new D(3.0,2.0);
foo.print();
```

ii)
```
D bar = new D();
bar.print(1 + " ");
```

iii)
```
A apple = new A();
apple.print();
```

iv)
```
A grannySmith = new B();
grannySmith.fax();
```

v)
```
A redDelicious = new D();
C pear = (D)redDelicious;
pear.print();
```

vi)
```
A peach = new D();
B banana = (B) peach;
banana.print();
```

vii)
```
A strawberry = new D();
((D)strawberry).print(1.0,2.0);
```

viii)
```
A blackberry = new D();
B latte = (B) blackberry;
latte.fax();
```

(8 poäng)

Uppgift 2.

Betrakta nedanstående klass:

```
public class Zoo {
   public enum Animal {DOG, CAT, BIRD}
   public void feedTheAnimal(Animal theAnimal) {
      switch (theAnimal) {
         case DOG:
            // Complex code special for Dog
            break;
         case CAT:
            // Complex code special for Cat
            break;
         case BIRD:
            // Complex code special for Bird
      }
   }//feedTheAnimal

   //code not shown here

}//Zoo
```

a) Klassen bryter mot en grundläggande designprincip. Vilken? Förklara vad principen innebär och på vilket sätt klassen Zoo bryter mot principen.

b) Åtgärda problemet. Förklara din lösning och visa den i Java-kod. (2 + 5 poäng)

Uppgift 3.

Vad skriver koden nedan ut? IllegalArgumentException och NullPointerException är båda subklasser till RuntimeException.

```
public static void main(String[] args) {
   try {
      doIt();
      System.out.println("main after doIt");
      doOther();
      System.out.println("main after doOther");
   } catch (IllegalArgumentException e) {
      System.out.println("main IllegalArgumentException");
   } catch (RuntimeException e) {
      System.out.println("main RuntimeException");
   }
}

public static void doIt() {
   System.out.println("enter doIt");
   try {
      doOther();
   } catch (IllegalArgumentException e) {
      System.out.println("doIt IllegalArgumentException");
      throw new NullPointerException();
   } catch (NullPointerException e) {
      System.out.println("doIt NullPointerException");
      throw new NullPointerException();
   } finally {
      System.out.println("doIt finally");
   }
   System.out.println("exit doIt");
}

public static void doOther() {
   System.out.println("enter doOther");
   throw new IllegalArgumentException();
}
```

(5 poäng)

Uppgift 4.

Betrakta nedanstående specifikationer för metoden

```
public static void addScore(String name, List<Double> scores, Map<String, Double> gradeBook)
```

S1)
```
@requires name != null and scores != null and gradeBook != null
@modifies gradeBook
@effects add a mapping <name, overallScore> to gradeBook
```

S2)
```
@requires name != null and scores != null
@modifies gradeBook
@effects add a mapping <name, overallScore> to gradeBook
@throws IllegalArgumentException if gradeBook is null
```

a) Är någon av de båda specifikationerna starkare än den andra? Ange i så fall vilken. Motivera ditt svar!

b) Nedan ges en möjlig implementation av metoden addScore:

```
public static void addScore(String name, List<Double> scores, Map<String, Double> gradeBook) {
   if (name == null || scores == null || gradeBook == null)
      throw new IllegalArgumentException();
   double grade = 0.0;
   for (double s : scores)
      grade = grade + s;
   if (scores.size() > 0)
      grade = grade / scores.size();
   gradeBook.put(name, grade);
}//addScore
```

i) Uppfyller denna implementation specifikationen S1 ovan? Motivera ditt svar!

ii) Uppfyller denna implementation specifikationen S2 ovan? Motivera ditt svar!

c) Ovanstående implementation bryter mot en grundläggande designprincip. Vilken?

(2+2+2 poäng)

Uppgift 5.

Antag att vi har klassen Movie nedan, för att representera filmer:

```
public class Movie {
   private String title;      // movie title
   private int year;          // year released
   private int minutes;       // length of the movie
   ...
} //Movie
```

a) Skriv en lämplig equals-metod för klassen. Två objekt av klassen Movie skall betraktas som lika om de har samma titel och samma utgivningsår.

b) Nedan ges fyra möjliga implementationer av metoden hashCode för klassen Movie. Vilka av dessa är korrekta? Vilken implementation är att föredra? Motivera dina svar!

i)
```
public int hashCode() {
   return 331;
}
```

ii)
```
public int hashCode() {
   return title.hashCode();
}
```

iii)
```
public int hashCode() {
   return title.hashCode() + minutes;
}
```

iv)
```
public int hashCode() {
   return title.hashCode() + year;
}
```

(3 + 3 poäng)

Uppgift 6.

Betrakta klasserna Bee och Flower nedan

```
public class Bee {
   private final String name;

   public Bee(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
   //. . .
}//Bee

public class Flower {
   private final String name;

   public Flower(String name) {
      this.name = name;
   }

   public void visit(Bee b) {
      System.out.println(name + " is being visited by " + b.getName());
   }

   public void bloom() {
      System.out.println(name + " is blooming!");
   }
   //. . .
}//Flower
```

Komplettera dessa klasser så att designmönstret Observer realiseras (genom att använda Observable och Observer eller genom att använda PropertyChangeSupport och PropertyChangeListener). När en blomma blommar meddelar den detta till alla bin som är observatörer, och när ett bi som får reda på att en blomma blommar besöker biet blomman. Nedanstående huvudprogram illustrerar hur det hela är tänkt att fungera:

```
public static void main(String[] args) {
   Flower f = new Flower("Blossom");
   Bee b1 = new Bee("Bob");
   Bee b2 = new Bee("Fred");
   f.addObserver(b1);
   f.addObserver(b2);
   f.bloom();
}//main
```

Utskriften blir:

```
Blossom is blooming!
Blossom is being visited by Fred
Blossom is being visited by Bob
```

(6 poäng)

Uppgift 7.

Betrakta följande klass:

```
import java.util.*;
public class CrossReferenceList {
   private Map<String, Set<Integer>> map;

   public CrossReferenceList() {
      //skall implementeras i deluppgift a)
   }

   public void add(String word, int sideNbr) {
      //skall implementeras i deluppgift b)
   }

   public String getPageNumbersOfWord(String word) {
      //skall implementeras i deluppgift c)
   }

   public String getReferenceListAsString() {
      //skall implementeras i deluppgift d)
   }
} //CrossReferenceList
```

Klassen CrossReferenceList används för att skapa ett register (en korsreferenslista) till en bok, där orden i registret skall skrivas ut i alfabetisk ordning och där sidnumren för varje ord ska vara ordnade i stigande följd.

Exempel:

```
class 3, 5, 7, 9
int 6
interface 5, 19
java 1, 2, 5, 8, 19
object 2, 6, 7, 9
```

Även om samma ord finns flera gånger på samma sida ska sidnumret bara förekomma en gång.

I implementationen ovan implementeras registret med hjälp av datatypen Map, med orden som nycklar och en mängd (Set) med sidnummer som värde.

a) Implementera konstruktorn som skapar en tom Map.

b) Implementera metoden add, som uppdaterar registret med ordet word på sidan sideNr.

c) Implementera metoden getPageNumbersOfWord, som returnerar en sträng som innehållet ordet word följt av de sidnummer som ordet förekommer på (dvs. en sträng som representerar en rad i exemplet ovan)..

d) Implementera metoden getReferenceListAsString, som returnerar en sträng som representerar hela registret (dvs. samtliga rader i exemplet ovan).

(1+3+3+2 poäng)

Uppgift 8.

a) Skriv en *trådsäker klass* Counter som representerar en heltalsräknare. Klassen skall innehålla en konstruktor som sätter räknaren till 0, samt metoderna

void add(**int** number)	som adderar number till räknaren
int getValue()	som returnerar värdet av räknaren

b) Skriv ett program som beräknar summan av elementen i ett heltalsfält genom att använda två trådar. Den ena tråden beräknar summan för elementen som finns i den första halvan av fältet, och den andra tråden beräknar summan för elementen som finns i den andra halvan av fältet. Den totala summan av samtliga element skrivs ut av huvudtråden som exekverar main-metoden. En instans av klassen Counter (från deluppgift a) används för att sköta kommunikationen mellan trådarna. Nedanstående (ofullständiga) main-metod visar hur det skall fungera:

```
import java.util.Arrays;
public class MainThread {
   public static void main(String[] arg) throws InterruptedException {
      int[] arr = {1,2,3,4,5,6,7,8,9,10, 11};
      int[] firstHalf = Arrays.copyOfRange(arr, 0, arr.length/2);          //första halvan av fältet
      int[] secondHalf =  Arrays.copyOfRange(arr, arr.length/2, arr.length);  //andra halvan av fältet
      Counter counter = new Counter();                                     //"kommunikationskanalen"
      ArrayAdder a1 = new ArrayAdder(firstHalf, counter);                  //a1 beräknar första halvan
      ArrayAdder a2 = new ArrayAdder(secondHalf, counter);                 //a2 beräknar andra halvan
      //kod som saknas, din uppgift att skriva
     } //main
} //MainThread
```

Färdigställ main-metoden och implementera klassen ArrayAdder. Det är ditt val om du låter klassen ArrayAdder utöka klassen Thread eller implementerar interfacet Runnable.

(3 + 6 poäng)

Uppgift 9.

Antag att följande klasser är givna:

```
public class ArtistTool { ... }
public class Brush extends ArtistTool { ... }
public class DetailBrush extends Brush { ... }
public class Pencil extends ArtistTool { ... }
```

Antag vidare att vi i ett program har gjort följande deklarationer:

```
ArtistTool t;
Brush b;
DetailBrush d;
Pencil p;
List<Brush> lb;
List<? extends Brush> leb;
List<? super Brush> lsb;
```

Ange för var och en av nedanstående satser om satsen är korrekt eller ger kompileringsfel. Svaren skall motiveras!

a) lb.add(d); b) lsb.add(d); c) leb.add(d); d) t = lsb.get(0);

e) t = leb.get(0); f) d = leb.get(0); g) lsb.add(b); h) leb.add(b);

(4 poäng)

Tentamen 150413 - LÖSNINGSFÖRSLAG

Uppgift 1.

i) Utskriften blir ”First print in D”. Både statisk och dynamisk typ är D. Klassen D överskuggar metoden print(), varför denna metod exekveras.

ii) Utskriften blir ”In the object”. Både statisk och dynamisk typ är D. Metoden print(String) finns inte i klassen D. Inte heller superkalssen C tillhandahåller denna metod, men dock metoden print(Object) och eftersom String är en subtyp till Object är det denna metod som exekveras.

iii) Kompileringsfel. A är ett interface och kan inte instansieras.

iv) Kompileringsfel. Den statiska typen är A och A definierar inte metoden fax.

v) Utskriften blir: ”First print in D”. A är supertyp till D, varför ett objekt av typen D kan tilldelas till en variabel av typen A. C är en supertyp till D, varför ett objekt av typen D kan tilldelas till en variabel av typen C. C definierar metoden print(). Den dynamiska typen vid exekveringen av pear.print() är D.

vi) Exekveringsfel: B och D är inkompatibla typer. Kompileringen lyckas eftersom explicit typomvandling är ett direktiv till kompilatorn att bortse typkontrollen.

vii) Utskriften blir ”Second print in D”.

viii) Exekveringsfel: B och D är inkompatibla typer. Kompileringen lyckas eftersom explicit typomvandling är ett direktiv till kompilatorn att bortse typkontrollen.

Uppgift 2.

a) Klassen bryter mot *Open Closed Principle* (OCP), som säger att en klass skall vara utvidgningar men stängd för förändringar. Klassen Zoo måste förändras om flera typer av djur skall läggas till.

b) Lösningen på problemet är att använda polyformism.

```
public abstract class Animal {
    abstract void feed();
}//Animal

public class Dog extends Animal {
    @Override
    public void feed(){
      // Complex code special for Dog
    }//feed
}//Dog

public class Cat extends Animal {
    @Override
    public void feed(){
      // Complex code special for Cat
    }//feed
}//Cat

public class Bird extends Animal {
    @Override
    public void doIt(){
      // Complex code special for Bird
    }//feed
}//Bird

public class Zoo {
    public void feedTheAnimals(Animal theAnimal){
        theAnimal.feed();
    }//feedTheAnimal

   //code not shown here

}//Zoo
```

Uppgift 3.

Utskriften blir:

```
enter doIt
enter doOther
doIt IllegalArgumentException
doIt finally
main RuntimeException
```

Uppgift 4.

a) Specifikation S2 är starkare, eftersom den har svagare förvillkor.

b) Implementationen uppfyller båda specifikationerna.

c) Implementationen bryter mot principen *Separation of Concern*. Metoden gör två saker: beräknar medelvärdet av score och lagrar denna information i datastrukturen. För att efterleva *Separation av Concern* skall beräkningen av medelvärdet göras i en egen metod.

```
public void addScore(String name, List<Double> scores, Map<String,Double> gradeBook) {
    if (name == null || scores == null || gradeBook == null)
        throw new IllegalArgumentException();
    double grade = averageGrade(scores);
    gradeBook.put(name,grade);
}//addScore

private double averageGrade( List<Double> scores) {
    double grade = 0.0;
    for (double s : scores)
        grade = grade + s;
    if (scores.size() > 0)
        grade = grade / scores.size();
    return grade;
}//averageGrade
```

Uppgift 5.

a)

```
@Override
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null)
        return false;
    if (this.getClass() != o.getClass())
        return false;
    Movie other = (Movie) o;
    return this.title.equals(other.title) && this.year == other.year;
} //equals
```

Även nedanstående implementation accepteras:

```
@Override
public boolean equals(Object o) {
    if (! (o instanceof Movie)) {
        return false;
    }
    Movie other = (Movie) o;
    return this.title.equals(other.title) && this.year == other.year;
} //equals
```

b) Implementation i), ii) och iv) är korrkta. Implementation iii) felaktig eftersom den använder instansvariabeln minutes vilket inte equals-metoden gör. Alternativ iv) är att föredra då den ger bäst spridning av hashkoderna, eftersom både title och year används vid beräkningen. Alternativ i) ger att alla instanser får samma hashkod, och alternativ iii) ger att alla instanser med samma titel får samma hashkod.

Uppgift 6.

Lösning med Observer och Observable:

```
import java.util.Observer;
import java.util.Observable;
public class Bee implements Observer {
    private final String name;
    public Bee(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void update(Observable obs, Object obj) {
        if (obs instanceof Flower) {
            Flower f = (Flower) obs;
            f.visit(this);
        }
    }
}//Bee
```

```
import java.util.Observable;
public class Flower extends Observable {
    private final String name;
        public Flower(String name) {
            this.name = name;
        }
        public void visit(Bee b) {
            System.out.println(name + " is being visited by " + b.getName());
        }
        public void bloom() {
            System.out.println(name + " is blooming!");
            setChanged();
            notifyObservers();
        }
}//Flower
```

Lösning med PropertyChangeSupport och PropertyChangeListener:

Om inga förändringar skall göras i main-metoden i uppgiften måste vi införa ett interface:

```
import java.beans.PropertyChangeListener;
public interface IObservable {
    void addObserver(PropertyChangeListener observer);
    void removeObserver(PropertyChangeListener observer);
}//IObserver

import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class Flower implements IObservable {
    private PropertyChangeSupport pcs = new PropertyChangeSupport(this);
    private final String name;
    public Flower(String name) {
        this.name = name;
    }
    public void addObserver(PropertyChangeListener observer) {
        pcs.addPropertyChangeListener(observer);
    }
    public void removeObserver(PropertyChangeListener observer) {
        pcs.removePropertyChangeListener(observer);
    }
    public void visit(Bee b) {
        System.out.println(name + " is being visited by " + b.getName());
    }
    public void bloom() {
        System.out.println(name + " is blooming!");
        pcs.firePropertyChange("Some text", this, name); // eller några andra lämpliga parametrar
    }
}//Flower

import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class Bee implements PropertyChangeListener {
    private String name;
    public Bee(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void propertyChange(PropertyChangeEvent ev) {
        if (ev.getSource() instanceof Flower) {
            Flower f = (Flower) ev.getSource();
            f.visit(this);
        }
    }
}//Bee
```

Uppgift 7.

```
import java.util.*;
public class CrossReferenceList {
   private Map<String, Set<Integer>> map;

   public CrossReferenceList() {
       map = new TreeMap<String, Set<Integer>>();
   } //constructor

   public void add(String word, int sideNbr) {
       Set<Integer> nbrs = map.get(word);
       if (nbrs == null) {
           nbrs = new TreeSet<Integer>();
           map.put(word, nbrs);
       }
       nbrs.add(sideNbr);
   } //add

   public String getPageNumbersOfWord(String word) {
       Set<Integer> nbrs = map.get(word);
       if (nbrs == null) {
           return word;
       }
       StringBuilder b = new StringBuilder();
       b.append(word);
       b.append(' ');
       Iterator<Integer> itr = nbrs.iterator();
       while (itr.hasNext()) {
           int i = itr.next();
           b.append(i);
           if (itr.hasNext()) {
               b.append(", ");
           }
       }
       return b.toString();
   } //getAsString

   public String getReferenceListAsString() {
       String b = "";
       Set<String> keys = map.keySet();
       for (String s : keys) {
           b = b +  getPageNumbersOfWord(s) + "\n";
       }
       return b;
   }// getReferenceListAsString
} //CrossReferenceList
```

Uppgift 8.

a)

```
public class Counter {
    private int value;
    public Counter() {
        value = 0;
    }//constructor

    public synchronized void add(int number) {
        value = value + number;
    }//add

    public int getValue() {
        return value;
    }//getValue
}//Counter
```

b)

Lösning med som utökar klassen Thread:

```
import java.util.Arrays;
public class MainThread {
    public static void main(String[] arg) throws InterruptedException {
        int[] arr = {1,2,3,4,5,6,7,8,9,10, 11};
        int[] firstHalf = Arrays.copyOfRange(arr, 0, arr.length/2);
        int[] secondHalf =  Arrays.copyOfRange(arr, arr.length/2, arr.length);
        Counter counter = new Counter();
        ArrayAdder a1 = new ArrayAdder(firstHalf, counter);
        ArrayAdder a2 = new ArrayAdder(secondHalf, counter);
        a1.start();
        a2.start();
        a1.join();
        a2.join();
        System.out.println(counter.getValue());
    } //main
} //MainThread

public class ArrayAdder extends Thread {
    private int[] arr;
    private Counter counter;

    public ArrayAdder(int[] arr, Counter counter) {
        this.arr = arr;
        this.counter = counter;
    } //constructor

    public void run() {
        int sum = 0;
        for (int i = 0; i < arr.length; i = i + 1) {
            sum = sum + arr[i];
        }
        counter.add(sum);
    } //run
}//ArrayAdder
```

Lösning med som inplementerar interfacet Runnable:

```
import java.util.Arrays;
public class MainThread {
    public static void main(String[] arg) throws InterruptedException {
        int[] arr = {1,2,3,4,5,6,7,8,9,10, 11};
        int[] firstHalf = Arrays.copyOfRange(arr, 0, arr.length/2);
        int[] secondHalf =  Arrays.copyOfRange(arr, arr.length/2, arr.length);
        Counter counter = new Counter();
        ArrayAdder a1 = new ArrayAdder(firstHalf, counter);
        ArrayAdder a2 = new ArrayAdder(secondHalf, counter);
        Thread t1 = new Thread(a1);
        Thread t2 = new Thread(a2);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println(counter.getValue());
     } //main
} //MainThread

public class ArrayAdder implements Runnable  {
    private int[] arr;
    private Counter counter;

    public ArrayAdder(int[] arr, Counter counter) {
        this.arr = arr;
        this.counter = counter;
    } //constructor

    public void run() {
        int sum = 0;
        for (int i = start; i < arr.length; i = i + step) {
            sum = sum + arr[i];
        }
        counter.add(sum);
    } //run
}//ArrayAdder
```

Uppgift 9.

Deklarationen

List<? **extends** Brush> leb;

innebär att leb kan referera till följande typer av listor: List<Brush>, List<DetailBrush> och List<OtherType>, där OtherType är en (för oss okänd) godtycklig subtyp till Brush.

Deklarationen

List<? **super** Brush> lsb;

innebär att lsb kan referera till följande typer av listor: List<Brush>, List<ArtistTool> och List<Object>.

a) Korrekt! DetailBrush är en subklass till Brush.

b) Korrekt! DetailBrush är en subklass till alla klasser som inkluderas i <? super Brush>, dvs ett element av typen DetailBrush kan läggas i listor av typerna List<Brush>, List<ArtistTool> och List<Object>, eftersom DetailBrush kär subtyp till Brush, ArtistTool och Object.

c) Kompileringsfel. DetailBrush är inte subklass till alla klasser som inkluderas i <? exdends Brush>. Antag att vi deklarerar en ny typ OtherType som är en subtyp till Brush. Det går det inte att lägga in ett objekt av DetailBrush i en lista av typen List<OtherType> eftersom DetailBrush inte är en subtyp till OtherType.

d) Kompileringsfel. ArtistTool är inte superklass till alla klasser som inkluderas i <? super Brush>. Eller mer specifikt: List<Object> inkluderas av <? super Brush> och ArtistTool är inte superklass till Object.

e) Korrekt. ArtistTool är superklass till alla klasser som inkluderas i <? extends Brush>. Eller mer specifikt: ett objekt av klassen ArtistTool kan kan läggas in i en lista som är av typen List<Bursh>, List<ArtistTool> eller List<Object>.

f) Kompileringsfel. DetailBrush är inte superklass till alla klasser som inkluderas i <? exdends Brush>. I en lista av typen List<Bursh>, kan elementen vara av typen Brush eller typen OtherType som är en, för oss okänd, subtyp till Brush.

g) Korrekt! Brush är subklass till alla klasser som inkluderas i <? super Brush>. , dvs ett objekt av typen Brush kan läggas in i en lista av typerna List<Bursh>, List<ArtistTool> och List<Object>.

h) Kompileringsfel. Brush är inte subklass till alla klasser som inkluderas i <? extends Brush>. Exempelvis kan inte ett objekt av typen Brush läggas in i en lista av typerna List<DetailBrush>.