Tentamen Programmering fortsättningskurs DIT950

Joachim von Hacht

Datum: 2015-03-17

Tid: 08.30-12.30

Hjälpmedel: Engelskt-Valfritt språk lexikon

Betygsgränser:

- U: -23
- G: 24-43
- VG: 44-60 (max 60)

Lärare: Joachim von Hacht. Någon besöker ca 10.00 och 11.30, 0707/311066

Granskning: Tentamen kan granskas på studieexpeditionen. Vi ev. åsikter om rättningen eposta mig och ange noggrant vad du anser är fel så återkommer jag (ta en bild och skicka).

Instruktioner:

- För full poäng krävs ett läsbart, begripligt och heltäckande svar. Generellt 1p för varje relevant aspekt av problemet, konkreta kodexempel kan ge mer.
- Det räcker med enbart relevanta kodavsnitt, övrig kod ersätts med "..." (aldrig import, hjälpklasser såsom Main, main-metod, etc....)
- Oprecisa eller alltför generella (vaga) svar ger inga poäng. Konkretisera och/eller ge exempel. Det är aldrig någon risk att vara övertydlig!

LYCKA TILL...

1. Vad avses med (förklara med en eller ett par meningar); 4p

   a) Imperativ programmering (jmf. gärna med funktionell programmering).

   b) Klassinvariant (class invariant).

   c) Implementationsarv.

   d) Statisk typ (compile time type).

2. Givet klasserna och gränssnitten nedan (ev. vänd sida). 6p

   a) Rita ett UML klassdiagram för dessa.

   b) Ange för a)-f) i koden nedan om kodstycket är korrekt eller ej. Om det ej är korrekt specificera om det är compile-time eller runtime-fel. Om det är korrekt, vad skrivs ut?

```
// a
C c = new A();
c.doIt();

// b
A a = new B();
a.doIt();

// c
IA ia = new C();
ia.doIt();

// d
IA a = new B();
IX x = (IX) a;
x.doOther();

// e
IX x = new C();
A a = x;
a.doIt();

// f
A a1 = new B();
a1.doYetOther(1);
```

```
// Types ----------------------------
public interface IA {
   public void doIt();
}

public interface IX {
   public void doOther();
}

public class A implements IA {
    public void doIt() {
       System.out.println("doIt A");
    }
    public void doYetOther(double i) {
       System.out.println("doYetOther A " + i);
    }
}

public class B extends A {
    public void doIt() {
       System.out.println("doIt B");
    }
    public void doYetOther(int d) {
       System.out.println("doYetOther B " + d);
    }
}

public class C extends A implements IX {
    public void doOther() {
       System.out.println("doOther C");
    }
}
```

3. Förklara utförligt begreppen värde- respektive referenssemantik (by value, by reference) och ge ett belysande kodexempel. Illustrera gärna med bilder. 4p

4. Klassen nedan bryter mot the Open Closed Principle (OPC). 6p

   a) Vad innebär OPC?

   b) På vilket sätt bryter klassen, eller kommer den att bryta, mot OPC?

   c) Åtgärda problemet. Förklara din lösning och visa den i Java-kod (skissa).

```
// The class
public class BreakOPC {
```

```
        public enum AnimalType { DOG, CAT, BIRD}
        public void feedTheAnimals(AnimalType animalType) {
          switch (animalType) {
            case DOG:
            // Complex code special for Dog
            break;
            case CAT:
            // Complex code special for Cat
            break;
            case BIRD:
            // Complex code special for Bird
            break;
            default: // General case
          }
        }
    }
```

5. Vad skriver koden nedan ut? IllegalArgumentException och NullPointerException är båda subklasser till RuntimeException. 6p

```
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();
    }
```

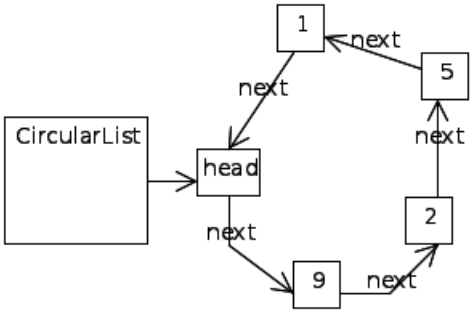
6. Designmönster har används en del i kursen. 8p

   a) Beskriv kort vad ett designmönster är.

   b) Redogör för två olika designmönster. Vad är syftet med mönstret (i vilket sammanhang/på vilket sätt används det)? Skissa struktur i UML och implementation i Java-kod.

7. Redogör för trådar (aktiva objekt). Varför används de (i denna kurs)? Vilka problem kan man få? Ge ett konkret kodexempel på något problem och hur det kan åtgärdas. 6p

8. Klassen nedan skall fungera som en enkellänkad cirkulär lista. En sådan ser ut som (vi antar att value är Integer); 12p



Man kan alltså bara gå åt ena hållet.

   a) Gör klart klassen genom att lägga till den kod som kan behövas vid kommentarerna (du får även ändra i befintlig kod). Tips: head används inte för någon data, jmf. root i Trie-klassen. Metoden add lägger till ett värde (och en nod) i listan. Exakt hur add skall implementeras får ni avgöra själva.

   b) Skapa en Iterator för klassen som en inre klass. Grässnittet för Iterator är;

```
public interface Iterator<T> {
    public boolean hasNext();
    public T next();
    public void remove(); // You don't need to implement remove
}
```

Du tar själv eventuella designbeslut. Vad skall hända om man modifierar listan samtidigt som man genomlöper den? Hantera detta.

c) Implementera clone-metoden för klassen. Clone behöver inte bibehålla ordningen på elementen.

```
/**
*  Class for a circular simple linked list
*/
public class CircularList<T> implements Iterable<T>, Cloneable {
    private Node head = new Node(null, null);
    private int size = 0;
    private class Node {
      T value;
      Node next;
      public Node(T value, Node next) {
        this.value = value;
        this.next = next;
      }
    }
    public CircularList() {
      // Possibly TODO
    }
    // Add element t to list
    public void add(T t) {
      // TODO
    }
    @Override
    public Iterator<T> iterator() {
      // TODO Iterator
    }
     // TODO clone
}
```

9. Antag att vi har typerna och koden nedan. Den markerade raden (<—) kommer inte att kompilera p.g.a. ett typfel. 8p

   a) Förklara varför.

   b) Modifiera metoden writeAll så att koden kompilerar d.v.s inte har något typfel. Du måste dessutom kort motivera varför det kommer att fungera.

```
// In some method
Sink<Object> s = ...;
Collection<String> cs = ...;
String str = ...;
str = writeAll(cs, s); <----- Will not compile

// Will send all elements to sink.flush and return last.
public static <T> T writeAll(Collection<T> coll, Sink<T> snk) {
    T last = null;
    for (T t : coll) {
        last = t;
        snk.flush(last);
    }
    return last;
}

// Interface
public interface Sink<T> {
    void flush(T t);
}
```