Tentamen Programmering fortsättningskurs DIT950

Joachim von Hacht

Datum: 2013-03-11

Tid: 14.00-18.00

Hjälpmedel: Engelskt-Valfritt språk lexikon

Betygsgränser:

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

Lärare: Joachim von Hacht. Någon besöker ca 15.00 och 16.30, 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 (ingen uttömmande förklaring krävs); 4p

   a) Tillstånd.

   b) Klassmetod (static).

   c) Icke muterbar.

   d) Runtime type.

2. Givet klasserna och gränssnitten nedan (ev. vänd sida). 8p

   a) Rita ett UML klassdiagram för dessa.

   b) Ange för a)-h) 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 (this.getClass().getSimpleName() ger klassens namn för aktuellt object t.ex. "C" eller "D")?

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

IY iy = new D();  // b
C c = (C) iy;
c.doOther();

A a = new B();  // c
a.doIt(1);

IX ix = new B(); // d
IY iy = new C();
ix = (IX) iy;
ix.doIt();

A a = new C(); // e
D d = (D) a;
d.doIt(1.0);

C c = new D(); // f
B b = (B) c;

C c = new C(); // g
A a = c;
a.doOther();

IY iy = new C(){  // h
    @Override
```

```
        public void doOther() { super.doOther() }
    };
    iy.doIt()

    // Types ----------------------------------
    public interface IX { public void doIt();}
        public interface IY { public void doOther();}
        public class A {

            public void doIt(double d){ System.out.println("Doit A " + d); }
        }
        public class B extends A implements IX{

            public void doIt(){System.out.println("Doit B");}
            public void doIt(int i){System.out.println("Doit B " + i);}
        }
        public class C extends A implements IY{

            public void doIt(){ System.out.println("Doit C");}
            public void doOther() { System.out.println("DoOther " +
              this.getClass().getSimpleName());}
            }
        public class D extends C {

            public void doIt(){ System.out.println("Doit D");}
        }
```

3. Beskriv hur objekt initieras i Java. 4p

4. Vad skriver programmet nedan ut? Ge en minutiös uttömmande förklaring till resultatet. Rita en bild som visar exakt hur många referenser (variabler) och hur många objekt som är inblandade vid metodanropet (swap) samt hur referenserna pekar. Ange också uttryckligen i text antal objekt = M och antal referenser = N. 6p

```
public class Wrapper {
    public Integer i;  // Bad with public
    public Wrapper(Integer i) { this.i = i; }
}
public class WrapperSwapper {
    public void swap(Wrapper v1, Wrapper v2) {
        Integer tmp = v1.i;
        v1.i = v2.i;
        v2.i = tmp;
```

```
    }
}
public static void main(String[] args) {
    WrapperSwapper ws = new WrapperSwapper();
    int a = 1;
    int b = 2;
    Wrapper v1 = new Wrapper(a);
    Wrapper v2 = new Wrapper(b);
    ws.swap(v1, v2);
    System.out.println("a= " + v1.i + " b= " + v2.i);
}
```

5. Gränssnitt (interface) är en viktig teknik och används i designprincipern; "Programming to an interface", "Interface segregation principle" och "Dependency inversion principle". Redogör för principerna och ge kodexempel (får gärna ge kodexempel från laborationerna). 6p

6. Implementationsarv är delvis problematiskt. Redogör för några problem (även specifikt utifrån Java). Ingen kod behövs. 4p

7. Redogör för två designmönster förutom Singleton. Vad är syftet? Skissa i Javakod hur de kan implementeras och visa strukturen m.h.a. ett UML (klass)diagram. 8p

8. Implementera en behållar som fungerar som en prioritetskö. I en prioritetskö läggs data i prioritetsordning (kön är ordnad efter prioritet). Behållaren skall använda en länkad datastruktur (d.v.s. noder). Allt som behövs skall finnas i en klass. Hantera eventuella undantag på bästa sätt. Ge klassen en så god design som möjligt. Följande publika metoder skall finnas 10p

```
// Insert data. Position in queue giving by prio
public void enqueue(int prio, Object data);
// Remove data with highest prio (first in queue)
public Object dequeue();
// Examples
q.enqueue(2, "aaa");    // 2    (queue looks like, data not shown)
q.enqueue(13, "vvv");   // 13,2
q.enqueue(99, "xxx");   // 99,13,2
q.enqueue(40, "eee");   // 99,40,13,2
q.dequeue();            // 40,13,2
q.dequeue();            // 13,2
```

9. Utifrån de givna gränssnitten nedan.

   a) Skapa en icke generisk klass MyFunction och en generisk klass MyStream som t.ex. kan användas enligt följande (tanken är att metoden map skall fungera som Haskells map). TIPS: Använd en List för elementen i Stream (List implementerar Iterable). 10p

```
// Example 1
MyStream<...> m = new MyStream(...);
Stream<...> s = m.map(new MyFunction());
Iterator<...> i = s.iterator();
while(i.hasNext()){
   System.out.println(i.next());
}
// Example 2
Stream<...> s = m.map(...).map(...).map(...);

// Interfaces
public interface Function<T, R> {
   // The function to apply
   R apply(T t);
}
public interface Stream<T> {

   // Apply function mapper to all elements and return new stream object
   // with result elements.
   public <R> Stream<R> map(Function<? super T, ? extends R> mapper);
   // Iterator over stream elements
   public Iterator<T> iterator();
}
```

b) Map metodens argument anges med typen Function<? super T, ? extends R>. Visa med ett kodexempel hur ? super T påverkar vila typer som kan användas TIPS: Skapa eget par av super sub-klass.