Tentamen Programmering fortsättningskurs DIT950

Joachim von Hacht

Datum: 2014-08-19

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 (ingen uttömmande förklaring krävs); 4p

   a) Objekt respektive klass.

   b) Primitiv respektive referensvariabel.

   c) Gränssnitt.

   d) Checked exception.

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?

```
public interface IA{ public void doA();}
public interface IX{ public void doX();}
public abstract class A implements IA {

    public void doA(){System.out.println("A.doA()");}
    public abstract void doC();
}
public class B extends A {

    public void doC(){ System.out.println("B.doC()");}
}
public class C extends B implements IX {

    public void doA(){ System.out.println("C.doA()");}
    public void doX(){ System.out.println("C.doX()");}
    public void doC(){ System.out.println("C.doC()");}
}
public class X implements IX {

    public void doX(){ System.out.println("X.doX()");}
    public void doA(){ System.out.println("X.doA()");}
}

public Main(){
    //a)
    B b = new B();
    b.doA();
    //b)
```

```
        IA a = new X();
        a.doA();
        //c)
        C c = new B();
        c.doC();
        //d)
        B b1 = new C();
        b1.doX();
        //e)
        IX x = new C();
        X x1 = (X) x;
        x1.doA();
        //f)
        IX x2 = new C();
        B b2 = (B) x2;
        b2.doC();
        //g)
        A a1 = new B();
        a1.doA();
        //h)
        IA a2 = new A();
        a2.doA();
    }
```

3. Vad innebär "Open Closed Principle" (OPC) och "Single Responibility Principle" (SRP). Förklara! 4p

4. Redogör för två olika designmönster förutom Singleton. Vilka problem vill man lösa? På vilket sätt löser designmönstret problemet. Ge en skiss i UML och relevant kod i Java för varje mönster. 6p

5. Ange för varje rad markerad 1-6 i metoden main nedan om raden;

   a) inte går att kompilera.

   b) ger en varning.

   c) kommer att orsaka ett undantag vid exekvering.

   d) kompilerar och exekverar utan problem. 6p

```
class Animal {}
class Cat extends Animal {}
class Dog extends Animal {}
public class AnimalHouse<E> {
    private E animal;
    public void setAnimal(E x) {animal = x;}
```

```
    public E getAnimal() { return animal; }
    static public void main(String[] args) {

        AnimalHouse<Dog> house1 = new AnimalHouse<Animal>(); // 1
        AnimalHouse<Animal> house2 = new AnimalHouse<Cat>(); // 2
        AnimalHouse house3 = new AnimalHouse(); // 3
        house3.setAnimal(new Dog()); // 4
        AnimalHouse<?> house4 = new AnimalHouse<Cat>(); // 5
        house4.setAnimal(new Cat()); // 6
    }
}
```

6. Att ett program har tillstånd medför problem. Redogör för tre olika sätt att begränsa problemen (begränsa tillståndet). Ge dessutom ett kort kodexempel som visar hur man implementerar respektive begränsning. 6p

7. Koden nedan ger en utskrift liknande (slumpmässiga 1:or); 6p

```
1010100000
0001010000
0000101010
0000000101
0000000010
```

Vi vill ha en utskrift;

```
0000000000
0000000000
0000000000
0000000000
0000000000
```

a) Förklara vad som är orsaken att den första utskriften innehåller 1:or (vad är problemet)?

b) Modifiera koden så att vi får utskriften med bara nollor. Förklara vad som gör att modifieringen ger det önskade resultatet.

8. En stack är en datastruktur som följer principen "last in first out" (LIFO). Strukturen definierar en position kallad "top". Två operationer "push" och "pop" lägger till respektive tar bort element från top. Effekten visas nedan. 10p

```
-        // [] Stack empty
push(1); // [1]  Underscore is top
push(2); // [2,1]
push(3)  // [3,2,1]
```

```
pop()    // [2,1]
pop()    // [1]
push(4)  // [4,1]
```

Implementera en klass Stack<T>. Stacken skall vara generisk och använda en länkad datastruktur för att spara värden. Den länkade strukturen skall byggas upp m.h.a. en inre Element-klass.

Följande operationer skall finnas;

```
// Enligt ovan
public void push(T t);
// Enligt ovan (obs värdet som utdata)
public T pop();
// Antal element i stacken
public int size();
// Flytta top till sist positionen  [4,3,2,1] -> [3,2,1,4]
public void roll();
```

9. I Java bör clone()-metoden implementeras på ett visst sätt (ett visst idiom).

   a) Visa hur den bör implementeras.

   b) Vad händer om clone() använder sig av en konstruktor. Förklara!

   c) Ett alternativ till clone är en kopieringskonstruktor. Visa hur man implementerar en kopieringskonstruktor. 10p