```
/*
*
*  Lösningsförslag tentamen DIT950
*  Datum 150317
*
*/


/*
*  -1 -
*/
För samtliga gäller ,se föreläsningsanteckningar.

/*
* - 2 -
*/
(Diagram visas inte, kontakta mig för ev frågor)

    // a
    C c = new A();  // Compile! Sub assigned super
    c.doIt();

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

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

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

    // e
    IX x = new C();
    A a = x;  // Compile! IX not sub to A
    a.doIt();

    // f
    A a1 = new B();   // Ok
    a1.doYetOther(1); // "doYetOther A 1.0


/*
* - 3 -
*/
Se föreläsningsanteckningar.
Kan t.ex. visa m.h.a Integer/int i >=  Integer/int j
eller metodanrop (olika swapper exempel)

/*
* - 4 -
```

```
*/
a) Se föreläsningar
b) Switch satsen gör att klassen måste modifieras om fler djur läggs
till
c) Använd polymorfism t.ex.

public abstract class Animal {
    abstract void doIt();
}

// Same for all case in switch statement
public class Dog extends Animal {
    @Override
    public doIt(){
        // Complex code special for Dog
    }
}

public void feedTheAnimals(Animal animal){
      animal.doIt();  // Switch statement gone
 }

/*
* - 5 -
*/
Utskrift

enter doIt
enter doOther
doIt IllegalArgumentException
doIt finally
main RuntimeException

/*
* - 6 -
*/
Se föreläsningsanteckningar.

/*
* - 7 -
*/
Trådar använd i denna kurs då vi har metoder
som kan ta lång tid (eftersom alla anrop blockerar
så måste resten av programmet vänta, t.ex kan GUI:et
låsas).
Genom att använda trådar verkar programmet göra
flera saker samtidigt, användaren kan t.ex. uppleva
programmet som mer responsivt. I verkligheten (med
en processor) byter programmet mellan de olika
trådarna. GUI:et får då viss processort tid och
kan på så sätt reagera på användarens kommandon.

Ett problem med trådar är att de delar icke-lokala
variabler. Om flera trådar samtidigt skriver/läser
```

samma variabler kan slutresultatet bli odefinierat
(ogiltigt tillstånd).

Exempel: Se Color från övningar 6

Genom att skapa atomära
operationer mha synchronized garanterar vi att
en tråd får köra klart och att därmed att
tillståndet förblir korrekt. Dock kan vi råka
ut för sk deadlock om vi överanvänder synchronized
eftersom detta låser objektet (inga synchronized
metoder kan köras av andra trådar). Om tråd T1 håller
(låser) objekt A och tråd T2 objekt B samtidigt som
A gör anrop på B och omvänt så kommer programmet
att låsa sig.

```
/*
*  - 8 -
*/
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;
        }
        /* NOTE: Can't use clone
         * T after compilation is Object
         * Objects.clone protected
        public Node clone(){
            Node n = (Node) super.clone();
            n.value = value.clone();  // Bad
            n.next = null;
        }*/
    }

    public CircularList() {
        head.next = head; // If not this problems at iterator
    }

    public void add(T t) {
        Node n = new Node(t, head.next);
        head.next = n;
        size++;
    }

    @Override
    public Iterator<T> iterator() {
```

```
        return new Iterator<T>() {

            private final int expectedSize = size;
            private Node pos = head;

            @Override
            public boolean hasNext() {
                // return true is not good, see clone()
                return pos.next.value != null;
            }

            @Override
            public T next() {
                if (expectedSize != size) {
                    throw new ConcurrentModificationException();
                }
                if(!hasNext()){
                    throw new NoSuchElementException();
                }
                pos = pos.next;
                return pos.value;
            }

            @Override
            public void remove() {
                // TODO Auto-generated method stub
            }
        };
    }

    @Override
    public Object clone() {
        CircularList<T> copy;
        try {
            copy = (CircularList<T>) super.clone();
            copy.head = new Node(null, null);
            copy.head.next = copy.head;

            Iterator<T> i = this.iterator();
            while (i.hasNext()) {
                T t = i.next();
                copy.add(t);
            }
            return copy;
        } catch (CloneNotSupportedException e) {
            throw new InternalError();
        }
    }
}


/*
*  - 9 -
*/
```

a) och b)
Se vidare http://docs.oracle.com/javase/tutorial/extra/generics/morefun.html

```
public static <T> T writeAll(Collection<? extends T> coll, Sink<?
super T> snk) {
        T last = null;
        for (T t : coll) {
            last = t;
            snk.flush(last);
        }
        return last;
    }
}
```