```
/*
*
*  Lösningsförslag tentamen DIT950
*  Datum 140819
*
*/


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


/*
* - 2 -
*/
(Diagram not shown)
 //a) Ok, extends A ( A.doA() )
 //b) Compile, X doesn't implement IA
 //c) Compile, C not supertype
 //d) Compile, B doesn't implement IX
 //e) Cast error C can't be X (different branches)
 //f) Ok ( C.doC() )
 //g) Ok ( A.doA() )
 //h) Compile A abstract

/*
* - 3 -
*/
Se föreläsningsanteckningar.

/*
* - 4 -
*/
Se föreläsningsanteckningar.


/*
* - 5 -
*/
1) A Compile, Generics invariant
2) A Compile, Invariant
3) D
4) D, Raw type get non generic behaviour (Dog compatible with
Object)
5) B or D (trivial warning)
6) A, Generic behavior, can't add to unknown

/*
* - 6 -
*/
Se föreläsningsanteckningar.
```

```
/*
* - 7 -
*/
a) Metoden inc() hinner inte köras klart innan tråden avbryts
diff() kommer att köras "mitt i" inc().
b) (kan även ange synchronized(s) { ... } i Client.
public class Data {

    private int x = 0, y = 0;

    // MUST synchronize both methods
    public synchronized int diff() {
        return x - y;
    }

    public synchronized void inc() throws InterruptedException {
        x++;
        Thread.sleep(7);
        y++;
    }
}

/*
*  - 8 -
*/
public class Stack<T> {

    private class Elem {

        private Elem(T t) {
            this.t = t;
        }
        T t;
        Elem next;
    }

    private Elem top; // = new Elem();
    private int size;

    public void push(T t) {
        Elem e = new Elem(t);
        e.next = top;
        top = e;
        size++;
    }

    public T pop() {
        Elem e = top;
        top = e.next;
        size--;
        return e.t;
    }

    public void roll() {
```

```
        Elem pos = top;
        Elem prev = pos;
        while (pos != null) {
            prev = pos;
            pos = pos.next;
        }
        T t = pop();
        Elem e = new Elem(t);
        e.next = null;
        prev.next = new Elem(t);
    }

    public int size() {
      return size;
    }


    }

/*
*  - 9 -
*/
a)
// This is any base class
public class A implements Cloneable {

    private int i = 111;
    private Date date = new Date();

    @Override
    public Object clone() {
        try {
            A a = (A) super.clone();
            a.date = new Date(date.getTime());
            return a;
        } catch (CloneNotSupportedException e) {
            // .. something is very wrong
            throw new InternalError();
        }
    }
}

// Any subclass
public class B extends A {

    private int j = 222;

    @Override
    public Object clone() {  // No throws here
        Object o = super.clone();
        // Dynamic type is B !!!
        System.out.println("Inside B: super.clone type is " +
o.getClass());
        B b = (B) o;
```

```
        // No references here, just return
        return b;
    }

}

b) Vi får cast exception vid B b = (B) o;
c)
public interface ICopyable {
    public Object copyOf();
}

// Any superclass
public class A implements ICopyable {

  // Run before ctor
  private String[] aStrs;

  public A() {
  }

  // Copy ctor, OK deep copy
  protected A(A a) {
    aStrs = new String[a.aStrs.length];
    for (int i = 0; i < aStrs.length; i++) {
      aStrs[i] = a.aStrs[i];
    }
  }

  // Covariant return type
  @Override
  public A copyOf() {
          System.out.println("A copyOf, runtime type of this is : "
+ this.getClass().getName());
    // Will call protected ctor
    return new A(this);
  }

}

// Any subclass
public class B extends A {

    // Run before ctor
    private String[] bStrs;

    public B() {
    }

    // Copy ctor, OK deep copy
    protected B(B b) {
        super(b);
        bStrs = new String[b.bStrs.length];
        for (int i = 0; i < bStrs.length; i++) {
```

```
            bStrs[i] = b.bStrs[i];
        }
    }

    // Covariant return type
    @Override
    public B copyOf() {
         System.out.println("B copyOf, runtime type of this is : " +
this.getClass().getName());
        // Will call protected ctor
        return new B(this);
    }

}
```