Examenvragen geconstrueerd op basis van menselijk geheugen. Dit kan mogelijk fouten bevatten.

1.a) Geef grafisch het software-ontwikkelingsproces weer voor objectgeoriënteerde software, geef hierbij duidelijk grafisch weer welke fasen in parallel door verschillende teams uitgevoerd kunnen worden
1.b) Tijdens het software-ontwikkelingsproces zijn verschillende tests nodig. Geef aan welke soorten tests we onderscheiden, door wie ze worden uitgevoerd, en wat hun bedoeling is
1.c) Leg uit: Dependency Injection, Software Engineering, fout-isolatie en incrementele patronen

2.a) Een methode heeft parameters a en b, deze moeten beide in [-30, 35] liggen. Indien |a-b| > 2 wordt 0 teruggegeven. Indien a-b <= -2, dan wordt -1 teruggeven en als a-b >= 2 wordt 1 teruggegeven. Hoeveel testvectoren zijn er nodig voor blackbox testing?
2.b) Implementatie is gegeven, geef minimale testvectoren om knoopbedekking/padbedekking te realiseren. Argumenteer.

```
    public int method(int[] a) {
        int N = 10;
        if(a== null) return -128;
        if(a.length != N) return -128;
        int result = 0;
        for(int i = 0; i < a.length; ++i) {
            if(i < a.length-1 && a[i] == a[i+1]) {
                result += 2*a[i];
                ++i;
            } else {
                result += a[i];
            }
        }
        return result;
    }
```

3.a) Gegeven slechte code met veel constructoroproepen. Teken het UML-diagram waarin je de slechte code verhelpt mbv het Abstract Method-patroon
3.b) Verhelp hetzelfde probleem mbv een Service Locator
3.c) 4 probleemstellingen; welk patroon is geschikt voor elk probleem?
3.d) Systeem van vectoren, gebruik Visitor-patroon om functionaliteit toe te voegen.
3.e) Gebruik nu deze visitors om wat bewerkingen uit te voeren.

4.a) Wat is een Counting Semaphore, en waarvoor wordt deze gebruikt?
4.b) Gegeven onderstaande implementatie van Counting Semaphore, zorg dat deze veilig gebruikt kan worden door meerdere draden.

```
public class CountingSemaphore {
    private final int bounds ;
    private int signals = 0;

    public CountingSemaphore(int maxBound) {
        bounds = maxBound;
    }
```

```
    public void acquire() {
        while(signals == bounds);
        ++signals;
    }

    public void release() {
        while(signals == 0);
        --signals;
    }

    public int getSignals() {
        return signals;
    }
}
```

4.c) Vermijd ook het actief wachten bij deze implementatie.

4.d) Hoe kan onderstaande code deadlocks veroorzaken?

```
public class Test {
    public static void main(String[] args) {
        Lock lock = new Lock();
        Waiter w = new Waiter(lock);
        Notifier n = new Notifier(lock, w);
        w.start();
        n.start();
    }
}

class Lock {}

class Waiter extends Thread {
    private final Lock lock;
    private boolean condition = true;
    public Waiter(Lock lock) {
        this.lock = lock;
    }

    public void setCondition(boolean condition) {
        this.condition = condition;
    }

    public void run() {
        while(condition) {
            synchronized (lock) {
                try {
                    lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

class Notifier extends Thread {
    private final Lock lock;
```

```
    private final Waiter w;
    public Notifier(Lock lock, Waiter w) {
        this.lock = lock;
        this.w = w;

    }

    public void run() {
        synchronized (lock) {
            w.setCondition(false);
            lock.notifyAll();
        }
    }
}
```

4.e) Los het probleem van 4.d op.

5.a) Leg uit hoe het JavaFX component framework het Observerpatroon implementeert

5.b) Implementatie dynamische proxy: leg uit wat deze dynamische proxy doet

```
interface I {
    public void A(String a, String b);
    public void B(int a);
    public void C(double a, double b);
}

class Cl implements I {

    @Override
    public void A(String a, String b) {
        System.out.println("A(String, String)");
    }

    @Override
    public void B(int a) {
        System.out.println("B(int)");
    }

    @Override
    public void C(double a, double b) {
        System.out.println("A(double, double)");
    }
}

class ALProxy implements InvocationHandler {
    private final I subject;
    private final ArrayList<Object> arguments = new ArrayList<>();
    public ALProxy(I subject) {
        this.subject = subject;
    }

    @Override
    public Object invoke(Object o, Method method, Object[] args) throws
Throwable {
```

```
        if(method.getName().equals("toString")) {
            return arguments.toString();
        }

        for(Object arg: args) {
            arguments.add(arg);
        }

        return method.invoke(subject, args);
    }
}
```

5.c) Geef het sequentiediagram voor onderstaande code

```
public class Test {
    public static void main(String[] args) {
        I i = (I) Proxy.newProxyInstance(ALProxy.class.getClassLoader(), new
Class[] {I.class}, new ALProxy(new Cl()));

        i.A("aa", "cc");
        i.B(5);
        i.C(7.5, 89);
        System.out.println(i);
    }
}
```

5.d) Geef output van de code van 5.c