MC2 Aufgabenpool

1 Welche der folgenden Aussagen treffen in Bezug auf Algorithmen und Datenstrukturen zu?

Wahr		Begründung
x	Absicherung gegen schlechte Datenverteilung kann die Laufzeit erhöhen.	
x	Arrays sind sehr effizient wenn normale Arrayzugriffe ausreichen.	
	Bei unbekannter Datenverteilung gehen wir von Zufallsverteilung aus.	
	Ein AVL-Baum ist stets effizienter als ein einfacher Suchbaum.	Muss nicht sein
x	Lineare Listen sind einfach zu implementieren.	
x	Mergesort wird auf linearen Listen häufiger eingesetzt als Quicksort.	
	Hash-Tabellen eignen sich vor allem gut für geordnete Daten.	Nein für ungeordnete
x	Bäume verwenden wir meist nur wo andere Datenstrukturen nicht passen.	

2 Welche der folgenden Aussagen treffen auf die Ein- und Ausgabe über Streams in Java zu?

Wahr		Begründung
	'new FileWriter(s)' erzeugt einen gepufferten Stream.	Nein ungepuffert
x	'new FileWriter(s)' erzeugt einen ungepufferten Stream.	
	'new FileWriter(s)' wirft eine IOException wenn s schon existiert.	Nein
	Ausgaben über gepufferte Streams gehen direkt an das Betriebssystem.	Nein ungepufferte gehen direkt
x	Puffer können als Wrapper vor nichtgepufferte Streams gehängt werden.	
x	Viele Methoden von PrintStream werfen keine IOException.	
x	Die meisten Methoden von Writer werfen eine IOException.	
x	Streams vom Typ Reader wandeln die Kodierung automatisch um.	
x	Streams werden nach der Verwendung mittels close() geschlossen.	
x	Gepufferte Streams sind meist effizienter als ungepufferte.	
	Ab Java 7 schließen sich am Methodenende offene Streams automatisch.	

Wahr		Begründung
x	Streams vom Typ Writer wandeln die Kodierung automatisch um.	
	Streams vom Typ InputStream wandeln die Kodierung automatisch um.	Nein
	Ungepufferte Streams sind meist effizienter als gepufferte.	

3 Welche der folgenden Aussagen treffen auf Schleifen und Schleifeninvarianten zu?

Wahr		Begründung
x	Die Abbruchbedingung kann nicht Teil einer Schleifeninvariante sein.	Invariante ändert sich nicht und ist vor und nach der Schleife gleich --> es kann nicht die Abbruchbedingung sein weils sonst unendlich lang sein würde.
	Durch Schleifeninvarianten werden Schleifen im Debugger iterierbar.	
	Eine Schleifeninvariante beschreibt, was jede Iteration ändert.	
	Schleifeninvarianten garantieren den Fortschritt jeder Iteration.	
x	Schleifeninvarianten helfen dabei, Programme statisch zu verstehen.	
x	Schleifeninvarianten müssen auch vor und nach der Schleife gelten.	
	Schleifeninvarianten müssen vor jedem Methodenaufruf erfüllt sein.	Nein nur vor jedem Schleifenaufruf.
x	Termination muss unabhängig von Schleifeninvarianten geprüft werden.	
x	Schleifeninvarianten beschreiben, was Iterationen unverändert lassen.	

4 Welche der folgenden Aussagen treffen auf das Testen großer Programme zu?

Wahr		Begründung
	Anwender sind wegen möglicher Verfälschungen nicht einzubeziehen.	Doch genau das ist wichtig, da bei Beta-Tests bzw Tests mit usern viele unerwartete Dinge eintreten an die man davor nicht dachte.
x	Auch intensives Testen kann nicht alle Fehler aufdecken.	

Wahr		Begründung
x	Eine Code-Review hilft beim Auffinden von Fehlerursachen.	
x	Experten für Softwaresicherheit sollen einbezogen werden.	
	Debugger eignen sich zum Aufdecken fast aller Fehlerursachen.	
	Regressions-Tests müssen fast immer händisch durchgeführt werden.	Nein auch automatisch
	White-Box-Testen legt Testfälle vor der Implementierung fest.	Nein das währen die Grey-Box-Tests
x	White-Box-Testen leitet Testfälle aus der Implementierung ab.	
x	Black-Box-Testen leitet Testfälle aus Anwendungsfällen ab.	
x	Mit absichtlichen Fehlern ist die Qualität des Testens prüfbar.	
	Schnittstellen-Tests überprüfen die Benutzeroberfläche.	Nein die Interfaces
	Wiederholte Fehlerkorrektur führt rasch zu fehlerfreien Programmen.	Nope ziemlich sicher nie fehlerfrei

5

```
public int median(int[] a) {
    return a[a.length / 2];
}
```

Welche der folgenden Aussagen können (jede für sich) als Vor- bzw. Nachbedingungen dieser Methode sinnvoll sein?

Wahr		Begründung
x	Nachbedingung: Gibt einen Eintrag etwa in der Mitte von a zurück.	
	Nachbedingung: Setzt voraus, dass a mindestens einen Eintrag hat.	Wäre Vorbedingung
x	Nachbedingung: Wirft eine Exception wenn a.length == 0.	
	Nachbedingung: Halbiert die Länge von a.	Machts ja nicht
x	Nachbedingung: Lässt a unverändert.	
	Nachbedingung: a.length > 0.	ist Vorbedingung
x	Vorbedingung: a != null	
x	Vorbedingung: a ist absteigend sortiert.	
x	Vorbedingung: a ist aufsteigend sortiert.	
	Vorbedingung: Gibt den Median zurück wenn a sortiert ist.	ist Nachbedingung

Wahr		Begründung
	Vorbedingung: Greift auf einen Eintrag von a zu.	ist gar nix
	Vorbedingung: Wirft eine Exception wenn a == null.	ist Nachbedingung

6 Welche der folgenden Aussagen treffen auf Iteratoren in Java zu?

Wahr		Begründung
x	Lineare Listen sind meist einfacher iterierbar als Binärbäume.	
	Mehrere Iteratoren auf dem gleichen Objekt stören sich gegenseitig.	Nein das ist der Sinn, dass man ungestört voneinander iterieren kann.
x	Iterator-Implementierungen sind eigene Klassen.	
x	Iterator-Implementierungen bestimmen die Reihenfolge der Iterationen.	
	Iteratoren über Bäume sind häufig rekursiv implementiert.	Sollten eigentlich nicht so sein aber ist einfacher meiner Meinung nach
	Die Reihenfolge der Iterationen zeigt die Art der Datenstruktur.	Iteratoren egal auf welcher Datenstruktur können beliebige Reihenfolge geben.

7 Welche der folgenden Aussagen treffen auf die notwendige Überprüfung von Eingabedaten zu?

Wahr		Begründung
	Alle Parameter einer Methode müssen in der Methode überprüft werden.	Nein es gibt ja Vorbediungungen
	Die Validierung soll erst möglichst knapp vor der Ausgabe erfolgen.	Nein am Anfang
	Java-Objekte vom Typ Pattern lesen nur überprüfte Daten ein.	ist eine kompilierte Darstellung eines regulären Ausdrucks. Es hat selbst keine Funktionalität, um "Daten einzulesen" oder zu "überprüfen". Seine Aufgabe ist es, das Muster des regulären Ausdrucks zu definieren.
x	Plausibilitätsprüfungen sollen direkt nach der Eingabe erfolgen.	
x	Reparaturversuche nicht plausibler Daten können gefährlich sein.	man könnte mehr kaputt machen als man rettet
x	Unzureichende Prüfung kann z.B. zu einer SQL-Injection führen.	
	Nicht validierbare Daten reparieren wir direkt nach der Eingabe.	

Wahr		Begründung
x	Reguläre Ausdrücke können Prüfungen von Datenformaten vereinfachen.	
x	Alle Daten von außerhalb des Programms müssen überprüft werden.	
	Wir müssen die Korrektheit prüfen, nicht nur die Plausibilität.	Nein es reicht Plausibilität

8 Welche der folgenden Aussagen treffen auf Klassen und Interfaces im Java-Collections-Framework zu?

Wahr		Begründung
x	Einträge in einem `TreeSet<E>` sind sortiert (über Iterator sichtbar).	
x	`Set<E>` erweitert `Collection<E>`, verbietet aber mehrfache Einträge.	
	`Map<K,V>` implementiert `Set<K>` als balancierten binären Suchbaum.	Nein Map bildet Schlüssel auf Werte ab. Set speichert nur Werte und hat keine Extra Schlüssel
	`ArrayDeque<E>` erweitert `ArrayList<E>` um die Methoden von `Deque<E>`.	Nein Deque ist was eigenes
x	In eine `Queue<E>` kann man neben `add()` auch mittels `offer()` einfügen.	
x	`LinkedHashMap<K,V>` ist sowohl Hash-Tabelle als auch lineare Liste.	
x	`LinkedList<E>` implementiert `Deque<E>` als doppeltverkettete Liste.	
	`LinkedList<E>` implementiert `Set<E>` als einfachverkettete Liste.	Nein Set ist unabhängig davon
	Einfügen mittels add ist in einer `ArrayList<E>` besonders effizient.	Nein da man bei einer möglichen Vergrößerung dann alle Elemente in ein größeres Array rüberkopieren muss.
	Einträge in einem `HashSet<E>` sind sortiert (über Iterator sichtbar).	Nein Reihenfolge wie die Elemente in der DS sind ist abhängig vom `hashCode()`
x	`TreeMap<K, V>` ist ein balancierter binärer Suchbaum	

9 Welche der folgenden Aussagen treffen auf die Einhaltung von Zusicherungen entsprechend Design-by-Contract zu?

Wahr		Begründung
	Clients müssen für die Einhaltung von Invarianten sorgen.	Nein der Server
	Clients müssen für die Einhaltung von Nachbedingungen sorgen.	Nope die Methode
x	Clients können sich auf die Einhaltung von Nachbedingungen verlassen.	
	Invarianten müssen zu ausnahmslos jedem Zeitpunkt erfüllt sein.	Nein nur am Schleifen Start und am Schleifen Ende bzw am Input und dann beim Output, dazwischen kanns auch verletzt werden.
x	Server müssen für die Einhaltung ihrer Nachbedingungen sorgen.	
	Server müssen für die Einhaltung ihrer Vorbedingungen sorgen.	Dafür ist der Client zuständig
x	Server müssen für die Einhaltung ihrer Invarianten sorgen.	
x	Server können sich auf die Einhaltung ihrer Vorbedingungen verlassen.	Dafür ist der Client zuständig
	Server können sich auf die Einhaltung ihrer Nachbedingungen verlassen.	Nein dafür müssen sie Sorgen
	Aufgerufene Methoden müssen für die Einhaltung von Vorbedingungen sorgen.	Nein der Aufrufer muss sich darum kümmern
x	Aufgerufene Methoden können sich auf die Einhaltung der Vorbedingungen verlassen.	
	Aufrufer müssen für die Einhaltung von Nachbedingungen sorgen.	Nein nur das Aufgerufte

10 Welche der folgenden Hoare-Tripel gelten (für Anweisungen in Java)?

Wahr		Begründung
	`{true} x = y<z ? y : z; {x>=y}`	Hat nichts mit Verhältnis zwischen x und y zu tun
x	`{y>0} while (x>0) x--; {y>0}`	y ändert sich nicht
x	`{true} x=0; {x>=0}`	$x = 0 \implies x \geq 0$
	`{x>0} while (x>0) x--; {x>0}`	x wird dann genau = 0 sein
	`{true} x++; {x>=0}`	Wir wissen nicht was x davor war.
x	`{x>2} if (x<0) x--; {x>2}`	
	`{x>2} if (x>0) x--; {x>2}`	x kann dann am Ende auch genau 2 sein.
	`{y<=0} while (++x < 0) y--; {y<0}`	Wir wissen nichts über x
x	`{x==5} x++; {x==6}`	5+1=6

Wahr		Begründung
x	`{true} x = y>z ? y : z; {x>=y}`	weil entweder man hat z was größer ist oder man hat y was = y ist.
x	`{y<0} while (++y <= 0) x--; {y>0}`	
	`{x==6} x++; {x==5}`	6+1=5?

11 Welche der folgenden Aussagen treffen für Design-by-Contract zu?

Wahr		Begründung
x	Invarianten beziehen sich auf Objektzustände.	
	Es gilt nur das, was explizit in Zusicherungen steht.	Nein man kann auch von impliziten Dingen ausgehen
x	Auch ohne Zusicherungen muss gelten, was allgemein erwartet wird.	
x	Vorbedingungen beschreiben häufig Eigenschaften von Parametern.	
	Invarianten beschreiben häufig Eigenschaften von Methodenparametern.	Während Methoden (und insbesondere die Validierung ihrer Parameter durch Vorbedingungen) dazu beitragen müssen, Invarianten eines Objekts zu erhalten, beschreiben die Invarianten selbst primär den internen Zustand von Schleifen oder Objekten, nicht die Eigenschaften von Methodenparametern. Letztere sind die Domäne der Vorbedingungen.
	Invarianten beziehen sich auf einzelne Methoden.	
	Eine Vorbedingung darf im Untertyp stärker sein als im Obertyp.	Nein andersherum
	Eine Nachbedingung darf im Untertyp schwächer sein als im Obertyp.	
	Nachbedingungen beschreiben häufig Eigenschaften von Parametern.	Das sind Vorbedinungen
x	Vor Methodenaufrufen müssen alle Invarianten erfüllt sein.	
x	Nachbedingungen beziehen sich auf einzelne Methoden.	

Wahr		Begründung
x	Objektzustände dürfen nur geändert werden, wo dies erwartet wird.	
x	Vorbedingungen beziehen sich auf einzelne Methoden.	
	Eine Invariante darf im Untertyp schwächer sein als im Obertyp.	Muss gleich sein
	Objektzustände dürfen geändert werden, wo dies nicht explizit verboten ist.	Nein nur wo es erwartet wird.

12 Welche der folgenden Aussagen treffen auf gut gewählte Kommentare in Programmen zu?

Wahr		Begründung
	Invarianten sollen bei Deklarationen lokaler Variablen stehen.	
x	Vor Methoden stehen hauptsächlich Vor-und Nachbedingungen.	
	Hinweise zur Programmanwendung stehen meist bei Schleifenköpfen.	Nein am Anfang vom Programm
	Kommentare sollen sich nur auf die Implementierung konzentrieren.	Nein auch auf die Anwendung etc
x	In Methodenrümpfen stehen hauptsächlich Schleifeninvarianten.	
	Die meisten Kommentare beschreiben, wie Anweisungen zu lesen sind.	
x	Die meisten Kommentare stehen an Schnittstellen (Methoden, Klassen).	
	Besonders gute Programmstellen enthalten besonders viele Kommentare.	
x	Kommentare legen auch fest, wer wofür verantwortlich ist.	Siehe Vor und Nachbedingungen
	Schleifeninvarianten finden wir meist bei Variablendeklarationen.	Nein in Schleifenköpfen
x	Invarianten sollen bei Deklarationen von Objektvariablen stehen.	
x	Kommentare sollen aus Sicht der Anwendung geschrieben sein.	

13 Welche der folgenden Aussagen treffen auf Ausnahmen und Ausnahmebehandlungen in Java zu?

Wahr		Begründung
x	Das Ergebnis von getMessage() soll die Fehlerart konkretisieren.	Das ist die Nachricht die wir bei eigenen Exceptions mit `super(message)` ausgeben
	'Propagieren einer Ausnahme' bedeutet: 'Ausnahme wird abgefangen'.	Nein das heißt, dass eine Ausnahme, die in einer Methode auftritt und nicht in dieser Methode abgefangen (behandelt) wird, an den aufrufenden Code (den "Caller") weitergegeben wird.
x	Alle Ausnahmen sind vom Typ Throwable.	
	Überprüfte Ausnahmen sind vom Typ RuntimeException oder Error.	
x	Gibt es mehrere catch-Blöcke, wird der erste passende ausgeführt.	
x	Ausnahmen vom Typ IOException sind überprüft (checked).	
x	Ausnahmen vom Typ Error werden in der Regel nicht abgefangen.	

14

Welche der folgenden Aussagen sind gültige Schleifeninvarianten dieser Schleife?

```
int n=8, f=1, i=1;
while (i < n)
    f *= ++i;
```

Wahr		Begründung
	f >= n	
x	f >= i	
x	i >= 1 && i <= n	
	i > 0 && i < n	Nein nach der letzten Iteration ist $i = n$ und nicht mehr $i < n$
	f ist das Produkt aller Zahlen von 1 bis n, also `1*...*n`	Nein das stimmt erst am Ende
x	f ist das Produkt aller Zahlen von 1 bis i, also `1*...*i`	
x	n == 8	

15 Welche der folgenden Aussagen treffen auf die Qualität großer Programme und das zur Erreichung der geforderten Qualität nötige Qualitätsmanagement zu?

Wahr		Begründung
	Zur Effekivitätssteigerung sind Team-Besprechungen zu vermeiden.	crazy call
x	Qualitätsanforderungen sollen analysiert und dargestellt werden.	
	Misstrauen gegenüber Teammitgliedern kann die Qualität verbessern.	
	Alle Qualitätskriterien müssen als gleich wichtig gesehen werden.	Gibt wichtigere und weniger wichtige
x	Pair-Programming gilt als qualitätssteigernde Maßnahme.	(2 Leute sitzen am gleichen Gerät)
x	Konfigurierbarkeit kann die Brauchbarkeit des Programms verbessern.	
x	Maßnahmen zur Schulung des Qualitätsbewusstseins sind effektiv.	
x	Formale Korrektheitsbeweise sind für große Programme extrem aufwendig.	
	Abhängigkeiten zu anderer Software sollen möglichst stark sein.	Schwach
	Konfigurierbarkeit beeinflusst die Brauchbarkeit des Programms nicht.	Doch tuts
x	Vertrauen gegenüber Teammitgliedern kann die Qualität verbessern.	

16 q, r und s seien Objektreferenzen ungleich null. Welche der folgenden Bedingungen müssen für jede Implementierung der Methoden 'boolean equals(Object obj)' und 'int hashCode' in Java gelten?

Wahr		Begründung
x	Aus r.equals(s) folgt s.equals(r).	Symmetrie
x	Aus q.equals(r) und r.equals(s) folgt s.equals(q).	Transitivität
x	Aus !r.equals(s) folgt !s.equals(r).	Symmetrie
x	Aus r.equals(s) folgt r.hashCode() == s.hashCode.	
	Aus r.hashCode() == s.hashCode() folgt r.equals(s).	2 Elemente können aus Zufall den selben `hashCode` haben aber nicht gleich sein.