Aufgabe 1 (10 Punkte)

`A` und `B` seien definiert durch `interface A<T> {}` und `class B<T> implements A<T> {}`.
Bitte markieren Sie jedes Auswahlfeld, bei dem der links stehende Typ ein Untertyp des darüber stehenden Typs ist. Es können keines, eines oder mehrere Felder pro Zeile auszuwählen sein.

	`A<String>`	`A<? super String>`	`A<? extends String>`	`A<?>`
`A<Object>`	○	○	⊗	⊗
`B<Object>`	○	○	○	○
`A<String>`	⊗	⊗	⊗	⊗
`B<String>`	⊗	○	○	○
`B<?>`	○	○	○	⊗
`B<? super Integer>`	○	○	○	⊗
`B<? super String>`	⊗	⊗	○	⊗
`B<? extends String>`	⊗	⊗	⊗	⊗
`A<B<String>>`	○	○	○	⊗
`B<A<String>>`	○	○	○	⊗

Aufgabe 2 (10 Punkte)

Jede Zeile enthält eine Java-Methode, die den Parameter mit einem anderen deklarierten Typ zurückgibt. `Set` und `HashSet` seien aus `java.util` importiert. Bitte markieren Sie das Auswahlfeld „statisch“ wenn bereits der Compiler ein Problem im Zusammenhang mit der Typänderung meldet (Syntaxfehler oder „unchecked“-Warnung), sonst „dynamisch“ wenn das Laufzeitsystem bei Ausführung der Methode eine Ausnahme wegen der Typänderung auslösen kann, sonst „fehlerfrei“.

	statisch	dynamisch	fehlerfrei
`Object f(String x) {return x;}`	○	○	⊗
`String f(Object x) {return (String)x;}`	○	⊗	○
`<T> HashSet<T> f(Set<T> x) {return (HashSet<T>)x;}`	○	○	⊗
`<T> T f(Object x) {return (T)x;}`	○	○	⊗
`<T> Set f(HashSet<T> x) {return x;}`	○	○	⊗
`<T> HashSet<T> f(Set x) {return (HashSet<T>)x;}`	○	⊗	○
`<T> Set<Set<T>> f(Set<HashSet<T>> x) {return x;}`	⊗	○	○
`<T> Set<Set<T>> f(HashSet<Set<T>> x) {return x;}`	○	○	⊗
`<T> Set<Object[]> f(Set<T[]> x) {return x;}`	⊗	○	○
`Set[] f(HashSet[] x) {return x;}`	○	○	⊗

Aufgabe 3 (10 Punkte)

Bitte markieren Sie in jeder Zeile das eine Auswahlfeld, bei dem die links stehende Aussage am ehesten eine Eigenschaft der darüber stehenden Parametrisierungsform in Java oder AspectJ ist.

	Annotationen	Aspekte	Generizität
wird auch *parametrischer Polymorphismus* genannt	○	○	⊗
`before()`-Advices werden vor normalem Code ausgeführt	○	⊗	○
Information steht der gesamten Werkzeugkette zur Verfügung	⊗	○	○
zahlreiche Typen von *Pointcuts* werden unterschieden	○	⊗	○
binäre Methoden sind über rekursive Deklarationen ausdrückbar	○	○	⊗
„`..`“ steht für eine beliebige Anzahl jedes beliebigen Zeichens	○	○	⊗
fast ausschließlich für *Querschnittsfunktionalität* geeignet	○	⊗	○
mit `@Target` wird festgelegt, wo Information anheftbar ist	⊗	○	○
die Syntax ist an die zur Definition von Interfaces angelehnt	⊗	○	○
zur Laufzeit sind Daten über Objekte von `Class` zugreifbar			

Aufgabe 4 (10 Punkte)

Bitte markieren Sie jedes Auswahlfeld, bei dem der links stehende Typausdruck (mit Typnamen aus den Paketen `java.util.function` und `java.lang`) ein Typ des darüber stehenden Lambdas ist. Es können keines, eines oder mehrere Felder pro Zeile auszuwählen sein.

	`Long::max`	`(x,y)->x-y`	`x->y->x*y`	`x->x*x`	`()->0`
`BiFunction<Long,Long,Long>`	⊗	⊗	○	○	○
`BiFunction<Long,Long,Integer>`	○		○	○	○
`BiFunction<Integer,Long,Long>`	○		○	○	○
`LongUnaryOperator`	○	○	○	⊗	○
`Function<Long,LongUnaryOperator>`	○	○	⊗	○	○
`Function<Long,Function<Long,Long>>`		○	⊗	○	○
`BinaryOperator<Long>`	○	○	○	○	○
`LongBinaryOperator`	○	○	○	○	○
`IntSupplier`	○	○	○	○	⊗
`LongSupplier`	○	○	○	○	⊗

Aufgabe 5 (10 Punkte)

Bitte markieren Sie jedes Auswahlfeld, bei dem die links stehende Eigenschaft auf den darüber stehenden Methodenaufruf zutrifft. Es können keines, eines oder mehrere Felder pro Zeile auszuwählen sein.

	`wait()`	`notifyAll()`	`Thread.sleep(5)`
weckt alle im Synchronisationsobjekt wartenden Threads auf	○	⊗	○
weckt alle im gesamten System wartenden Threads auf	○	●	○
Verwendung in einem `synchronized`-Block ist gefährlich			⊗
ist nur innerhalb eines `synchronized`-Blocks verwendbar			○
gehört zu den grundlegenden Synchronisationsmechanismen	⊗	⊗	○
kann eine `InterruptedException` zurückgeben	⊗	○	⊗
suspendiert die Ausführung des aktuellen Threads	⊗	○	⊗
suspendiert die Ausführung auf unbestimmte Zeit	⊗	○	○
bei Rückkehr ist der Lock auf den aktuellen Thread gesetzt	⊗	○	○
gibt Lock des Synchronisationsobjekts vorübergehend frei	⊗	○	○

Aufgabe 6 (10 Punkte)

Bitte markieren Sie jedes Auswahlfeld, bei dem die links stehende Kommandozeile (in `bash` ausgeführt) die darüber stehende Auswirkung hat. Es können keines, eines oder mehrere Felder pro Zeile auszuwählen sein.

	mehrere Prozesse laufen gleichzeitig	Prozesse über Pipeline verbunden	Prozesse laufen im Hintergrund	Standardeingabe von `cat` umgeleitet	Standardausgabe von `cat` umgeleitet	Fehlerausgabe von `cat` umgeleitet
`cat < a | wc > b`	⊗	⊗	○	⊗	⊗	○
`cat a |& wc &>> b`	⊗	⊗	○	○	⊗	⊗
`cat < a |& wc &> b &`	⊗	⊗	⊗	⊗	⊗	⊗
`cat a || wc b`	○	○	○	○	○	○
`for i in * ; do ( cat $i > ../bak/$i & ) ; done`	⊗	○	⊗	○	⊗	○
`for i in * ; do cat $i \| wc > ../wc/$i ; done`	⊗	⊗	○	○	⊗	○
`if test 'cat a' = "a b c" ; then wc b ; else wc c ; fi`	○	○	○	○	○	○
`cat a ; wc a`	○	○	○	○	○	○
`cat a &> b && wc b`	○	○	○	○	⊗	⊗
`cat | wc`						

Aufgabe 7 (10 Punkte)

Bitte markieren Sie jedes Auswahlfeld, bei dem die links stehende Aussage eine Eigenschaft des darüber stehenden Entwurfsmusters ist. Es können keines, eines oder mehrere Felder pro Zeile auszuwählen sein.

	Decorator	Proxy	Iterator	Prototype	Factory-Method
führt zu vielen kleinen Objekten	⊗	○	○	○	○
unterstützt Umkehrung der Abhängigkeiten	○	○	○	○	⊗
ist erzeugendes Entwurfsmuster	○	○	○	⊗	⊗
ist Entwurfsmuster für Struktur					
hilft große Zahl an Klassen zu vermeiden	○	○	○	⊗	○
häufig als innere Klasse implementiert	○	○	⊗	○	○
Objektidentität ist damit unzuverlässig	⊗	○	○	○	○
beruht auf Delegation	⊗	⊗	○	○	○
oft große Anzahl an Unterklassen nötig	○	○	○	○	⊗
für oberflächliche Erweiterungen geeignet	⊗	○	○	○	○

Aufgabe 8 (10 Punkte)

Bitte markieren Sie in jeder Zeile das eine Auswahlfeld, bei dem die links stehende Aussage am ehesten eine Eigenschaft des darüber stehenden Entwurfsmusters ist.

	Decorator	Proxy	Iterator	Prototype	Factory-Method
kann mit kovarianten Problemen umgehen	○	○	○	○	⊗
es gibt externe und interne Varianten	○	○	⊗	○	○
mehrere gleichzeitige Abarbeitungen möglich	○	○	⊗	○	○
zyklische Strukturen bereiten Probleme	○	○	○	⊗	○
Verantwortlichkeiten wieder entziehbar	⊗	○	○	○	○
robuste Varianten werden bevorzugt	○	○	⊗	○	○
flache von tiefen Kopien unterschieden	○	○	○	⊗	○
führt zu parallelen Klassenhierarchien	○	○	○	○	⊗
schlecht geeignet für umfangreiche Objekte					
kein Proxy, aber gleiche Struktur möglich	⊗	○	○	○	○

Aufgabe 9 (10 Punkte)

Bitte markieren Sie in jeder Zeile das eine Auswahlfeld, bei dem die links stehende Aussage am ehesten eine Eigenschaft des darüber stehenden Entwurfsmusters ist.

	Decorator	Proxy	Iterator	Prototype	Factory-Method
wird auch *Cursor* genannt	○	○	⊗	○	○
wird auch *Wrapper* genannt	⊗	○	○	○	○
wird auch *Surrogate* genannt	○	⊗	○	○	○
wird auch *Virtual-Constructor* genannt	○	○	○	○	⊗
Smart-Reference ist eine Variante davon	○	⊗	○	○	○
Subject ist ein Bestandteil	○	⊗	○	○	○
Aggregate ist ein Bestandteil	○	○	⊗	○	○
Creator ist ein Bestandteil	○	○	○	○	⊗
Product ist ein Bestandteil	○	○	○	○	⊗
Component ist ein Bestandteil	⊗	○	○	○	○

Aufgabe 10 (10 Punkte)

Bitte markieren Sie jedes Auswahlfeld, bei dem die links stehende Aussage eine Eigenschaft des darüber stehenden Entwurfsmusters ist. Es können keines, eines oder mehrere Felder pro Zeile auszuwählen sein.

	Factory-Method	Template-Method	Visitor	Singleton
sehr viele Klassen können entstehen	⊗	○	○	○
sehr viele Objekte können entstehen	○	○	○	○
sehr viele Methoden können entstehen	○	○	⊗	○
Anzahl erzeugter Objekte kontrollierbar	○	○	○	⊗
verwandte Operationen zentral verwaltet	○	○	⊗	○
häufig als Anti-Pattern angesehen	○	○	○	⊗
mehrere Arten *primitiver Operationen*				
das Hollywood-Prinzip spielt eine Rolle	⊗	⊗	○	○
Element ist ein Bestandteil	○	○	⊗	○
verwendet häufig *Hooks*	⊗	⊗	○	○