Beispiel 5 (10 Punkte)

Beantworten Sie die nachfolgenden Fragen kurz aber prägnant:

a) Erklären Sie das Konzept der Datenkapselung (*Data encapsulation*).

b) Was ist ein Objekt im Sinne von objektorientiertem Programmieren?

c) Was versteht man unter „Patterns“? Erklären Sie den Observer-Pattern.

d) Erklären Sie das Prinzip des White-Box-Tests. Worin besteht der Unterschied zu Black-Box-Tests?

Gruppe B

ICT Institut für Computertechnik

Prüfungsprotokoll

	Familienname:
	Matrikelnummer: / **Kennzahl:**
	Lehrveranstaltung: Objektorientiertes Programmieren

Schriftliche Prüfung	**Datum: 18. Juni 2014** **Ort: AudiMax**
	Schriftliche Note: **Negative Note: die erforderliche Anzahl von ____ Punkten wurde nicht erreicht.**

Beispiel 1 (55 Punkte)

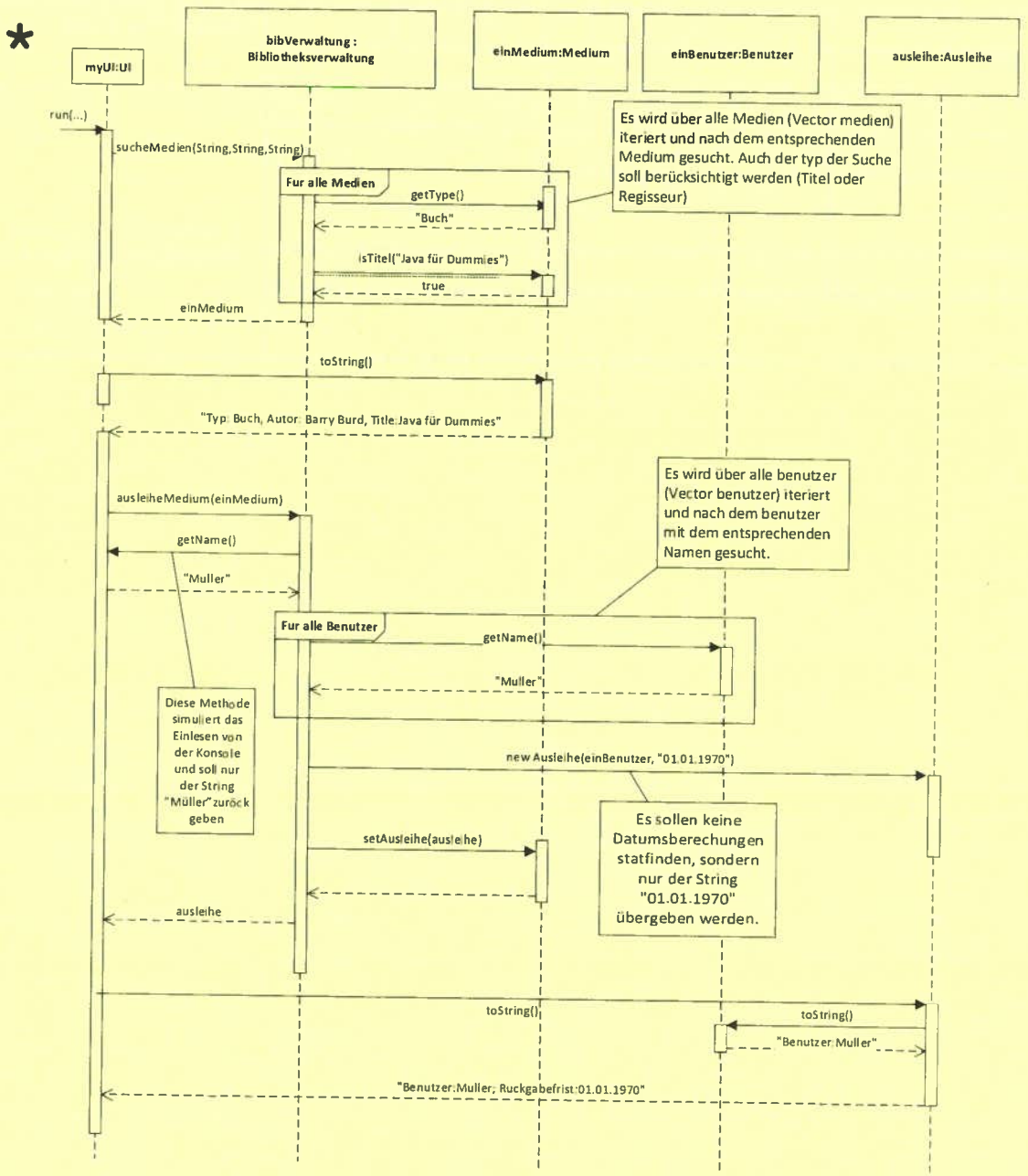
Gegeben sind ein **Klassen**- und ein **Sequenzdiagramm** eines Software-Entwurfs. Basierend auf diesen Diagrammen ist folgende Aufgabe zu lösen:

Implementieren Sie die Klassen (Methoden und Attribute) entsprechend dem Sequenz-, dem Klassendiagramm und dem angegebenen Code der Klasse "`Start`" (in dieser Klasse wird das Programm initialisiert).

Der Ablauf im Sequenzdiagramm (markiert mit "*") beginnt in der Methode `run` der Klasse `UI`.

Das Sequenzdiagramm beschreibt den Vorgang einer Medienausleihe in einer Bibliothek. Das Objekt der Klasse `UI` (Userinterface) ist die Schnittstelle zum Anwender. `Bibliotheksverwaltung` verwaltet eine Liste mit Medien, Bücher oder CDs (Vector `medien`), und sucht nach dem angeforderten Medium. Um ein Medium auszuleihen, muss der Benutzer zuerst nach dem Medium suchen, indem er den Typ des Mediums (Buch oder CD), Suchtyp (Titel oder Autor) und den Suchstring (z.B. „Java für Dummies“ für den Buchtitel) eingibt. Abhängig vom Suchtyp, soll im Programm die entsprechende Methode der Klasse Medium aufgerufen werden. Danach soll der Benutzer seinen Namen angeben. Nachdem der User seinen Namen angegeben hat, wird eine `Ausleihe` angelegt. Zuletzt werden die Daten der Ausleihe ausgelesen.

Sequenzdiagramm:



Beispiel 4 (10 Punkte)

Kreuzen Sie bei den folgenden zehn Fragen WAHR an, wenn die Aussage richtig ist, oder FALSCH, wenn die Aussage nicht richtig ist.

Bewertungsschema:

Für jede korrekt angekreuzte Aussage wird +1 Punkt gezählt. Wenn eine Frage nicht korrekt angekreuzt ist, wird 1 Punkt abgezogen (also –1). Wenn bei einer Frage weder WAHR noch FALSCH angekreuzt sind, gibt es 0 Punkte für diese Frage. Sie können in Summe bei Beispiel 4 nicht weniger als 0 Punkte haben, selbst wenn sich rein rechnerisch eine negative Punkteanzahl ergeben würde.

		WAHR	FALSCH
1)	Ein Blackbox-Test hat kein erwartetes Ergebnis, da dieses durch den Test bestimmt wird.	O	O
2)	Ein abstrakte Klasse kann auch mehr als ein Interface implementieren	O	O
3)	Ein Java Konstruktor hat nie den gleichen Namen wie die Klasse, zu der er gehört	O	O
4)	Ein Java Interface ist ein Datentyp.	O	O
5)	Ein Objekt im Sinne von OOP nimmt keinen Speicherplatz ein, da es nur ein abstraktes Konstrukt ist.	O	O
6)	Eine Subklasse erweitert eine andere Klasse.	O	O
7)	Eine Klasse erbt entweder den Zustand oder das Verhalten einer Superklasse.	O	O
8)	Eine Message (Nachricht an ein Objekt) enthält immer das empfangende Objekt, Namen der Methoden und mögliche Parameter.	O	O
9)	Ein Objekt ist eine Instanz einer Klasse.	O	O
10)	In Sequenzdiagrammen kann man das Senden von Nachrichten darstellen.	O	O

Beispiel 3 (10 Punkte)

Gegeben sind folgende Schnittstellen und Klassen:

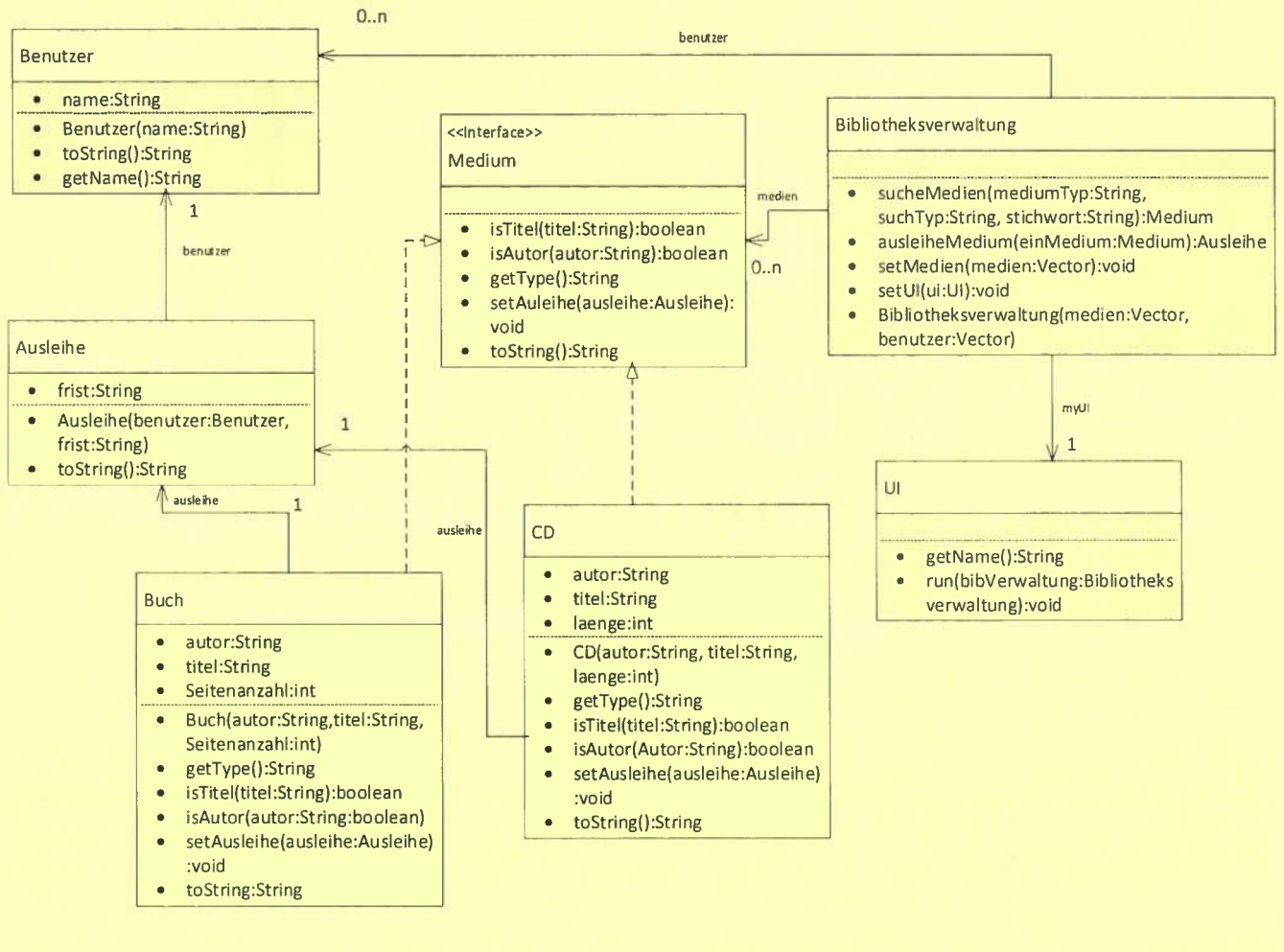
```
public interface Int_1 {
        ...
}

public interface Int_2 {
        ...
}

public abstract class AbstractC {
        ...
}

public class Class_1 {
        ...
}

public class Class_2 {
        ...
}
```

Kreuzen Sie **an**, ob die jeweilige Deklaration in Java **korrekt** ist:

	Ja	Nein
`public class BOUR extends Class_1, Class_2 { ... }`	☐	☐
`public class NEIGH implements Int_1 { ... }`	☐	☐
`public class SE extends Class_1 { ... }`	☐	☐
`public class DONT implements AbstractC { ... }`	☐	☐
`public class YFR implements Class_1 { ... }`	☐	☐
`public class COP extends Class_2 implements Int_1 { ... }`	☐	☐
`public class OMYO extends Int_1 { ... }`	☐	☐
`public class UR extends Int_1 implements AbstractC { ... }`	☐	☐
`public class EA extends AbsstractC { ... }`	☐	☐
`public class PL extends Int_2 implements Class_1 { ... }`	☐	☐

Klassendiagramm

Assoziationen mit einem Pfeil (gerichtete Assoziationen) im Klassendiagramm bedeuten, dass jene Klasse beim Pfeilursprung eine Referenz (gespeichert in einer Instanzvariable) zur Klasse am Pfeilende besitzt.



Klassen

```
import java.util.Vector;

public class Start {

	public static void main(String[] args) {
		Vector medien = new Vector();
		//Beispiel CD
		CD cd1 = new CD("Claude Nougaro", "Le Jazz et la Java", 100);
		medien.add(cd1);
		//Beispiel Buch
		Buch buch1 = new Buch("Barry Burd", "Java für Dummies", 352);
		medien.add(buch1);
		//Beispiel Benutzer
		Vector benutzer = new Vector();
		benutzer.add(new Benutzer("Müller"));
		benutzer.add(new Benutzer("Meier"));
		//erzeuge Bibliotheksverwaltung
		Bibliotheksverwaltung bibVerwaltung = new Bibliotheksverwaltung(medien,
benutzer);
		//erzeuge UI
		UI myUI = new UI();
		bibVerwaltung.setUI(myUI);
		//start Programm

		myUI.run(bibVerwaltung); //  *  Beginn des Ablaufes des Sequenzdiagrammes
		                         //     in der Methode "run" der Klasse "UI"
	}
}
```

Implementieren Sie hier die notwendigen Klassen:

Konsolenausgabe:

```
Lets do this!
val: 60
Almost TRUE
not true
Summer is AMAZING
```

Tabelle 1: Programmcodesnippets

Platzhalter	Lösungsvorschläge			
q1		Lets do this!	"Lets do this!"	„"
q2	"60"		void	60
q3	"wahr"	"TRUE"	FALSE	1
q4		"wahr"	false	"false"
q5		a = b(TRUE);	a = b;	a = new b;
q6	System.out.println("AMAZING")	show("AMAZING")	b.show("AMAZING")	a.show("AMAZING")
q7		"wahr"	"null"	"almost"
q8	val+ "val: "		"val: "+val	
q9		"Summer is "	"not true"	false
q10		extends childishInterface	implements childishInterface	overides parentAbstract
q11	void main public static	public private static main	public static void main	class main
q12	parentAbstract	Parent	Child	childishInterface
q13	Child	Child.show	System.out	Parent
q14	a	(Child) a	(Parent) a	a(Parent)
q15	implements childishInterface	inherits childishInterface	extends childishInterface	

Beispiel 2 (15 Punkte)

Gegeben ist ein unvollständiger Programmcode mit Platzhaltern (**q1** bis **q15**) und die Ausgabe des vollständigen Programms auf der Konsole.
Ihre Aufgabe ist es die fehlenden Stücke des Programmcodes zu ergänzen (um es leichter zu machen, sind in Tabelle 1 Lösungsvorschläge gegeben). Um dies zu tun markieren Sie in dieser Tabelle für jeden Platzhalter **q*X*** jenen Code-Teil, der zu einem korrekten Programm im Sinne der gewünschten Ausgabe auf der Konsole führt bzw., falls keine der angebotenen Lösungen passt, ergänzen Sie im entsprechenden leeren Felder eigenen Code. Bitte markieren Sie die von Ihnen gewählten Programmcodesnippets in Tabelle 1 durch Einkreisen.

```
public class Exam {
    q11(String[] args) {
        Parent a = new q12();
        Child b = new q13();

    System.out.println( q1 );

          a.show( q2 );
          a.show( q3 );
          b.show( q4 );
          q5
          a.q6;
          child c=q14;
 }
}
```

```
public class Parent {
  public void show( String val )
{
   System.out.println( "Almost "
+ val );}
   public void show( int val ) {
          if ( val == 0 )
     System.out.println( "q7" );
          else
     System.out.println( q8 );
  }
}
```

```
public class Child extends Parent q10 q15{
    public void show( String val ) {
         System.out.println( "Summer is " + val );
     }
     public void show( boolean val ) {
          if ( ! val )
               System.out.println( q9 );
          else
               System.out.println( "Summer is amazing" );
     }
     public int badKid( ) {
          return 60;
     }
}
```

```
public interface childishInterface {
    public int badKid();
}
```

```
public abstract class parentAbstract {
    abstract void amazing();
}
```


Implementieren Sie hier die notwendigen Klassen:

Implementieren Sie hier die notwendigen Klassen:

Implementieren Sie hier die notwendigen Klassen: