



Klausur

Studiengang:	Wirtschaftsinformatik
Jahrgang:	2016
Modul:	Programmierung und Programmiertechniken
Veranstaltung:	Fortgeschrittene Programmierung, Algorithmen und Datenstrukturen
Prüfer/-in:	Daniel Appenmaier
Datum:	29.06.2017
Bearbeitungszeit:	100 Minuten
Max. Punktzahl:	100 Punkte (+16 Zusatzpunkte)
Hilfsmittel:	Taschenrechner (nicht programmierbar) und Beiblatt zur Klausur
Sonstige Hinweise:	<ul style="list-style-type: none">• Lesen Sie die Aufgabenbeschreibung sorgfältig durch und achten Sie darauf, Antworten möglichst kurz und prägnant zu formulieren!• Benötigte Klassen- oder Schnittstellen-Imports müssen von Ihnen nicht explizit angegeben werden!• Schreiben Sie Ihre Antworten und Ihr Coding ausschließlich auf die Aufgabenblätter! Notfalls können Sie auch die Rückseite verwenden!

Matrikelnummer: _____ (hier eintragen!)

Viel Erfolg!

Durch Prüfer/-in auszufüllen:

Aufgabe	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
Maximale Punktzahl	12	6	2	6	4	3	8	3	4
Erreichte Punktzahl									

Aufgabe	2.1	2.2	2.3	2.4	2.5	2.6	2.7	Σ
Maximale Punktzahl	4	14	4	12	8	12	14	116
Erreichte Punktzahl								

Teil 1: Theorie (48 Punkte)

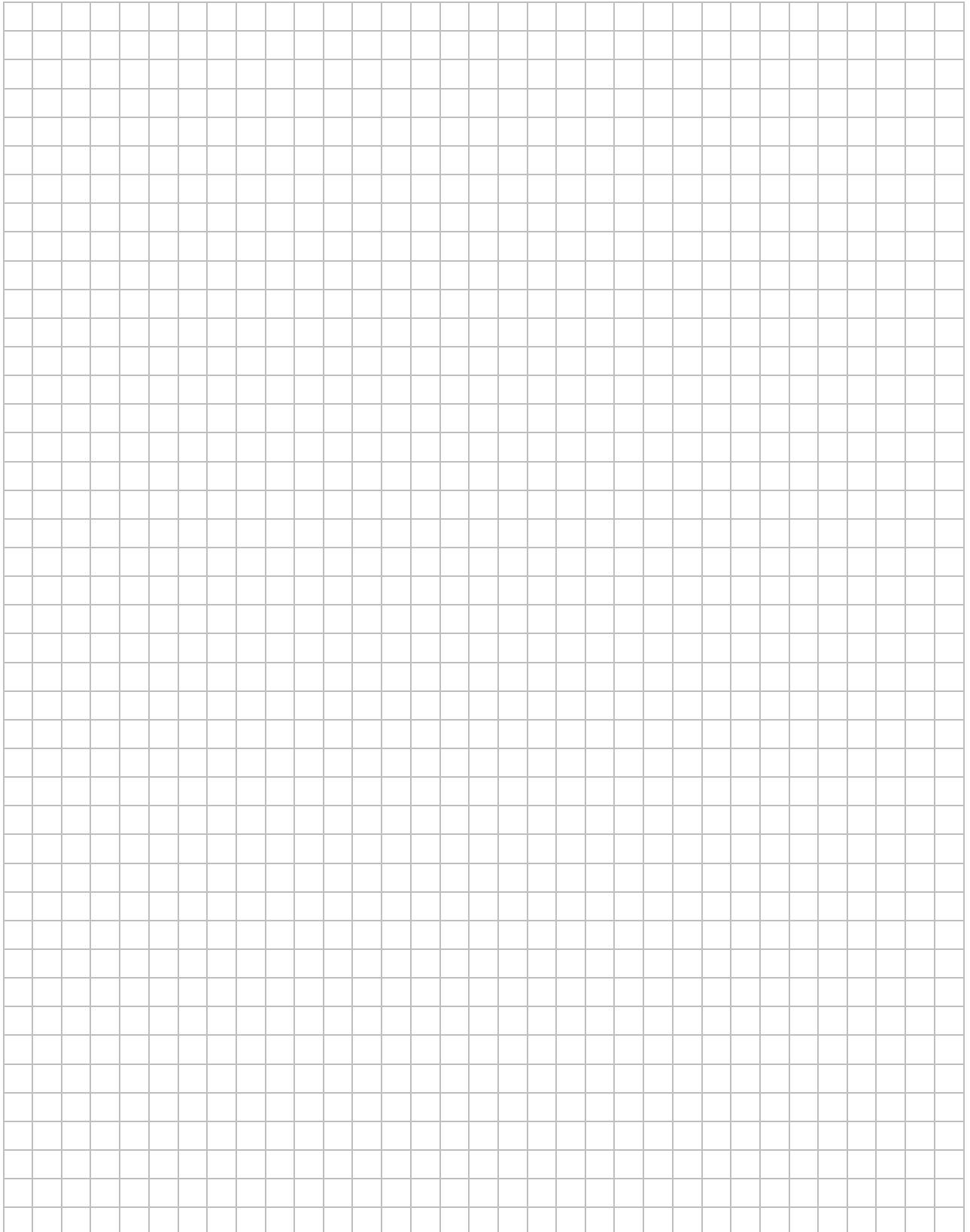
Aufgabe 1.1 (12 Punkte)

Kreuzen Sie an, welche der folgenden Aussagen richtig sind und welche falsch.
Hinweis: Falsche Antworten geben einen Minuspunkt!

Aussage	Richtig	Falsch
Die generische Klasse <i>ShoppingCart</i> könnte mit dem Datentyp <i>Service</i> deklariert werden.		
Der Konstruktor der Klasse <i>Service</i> ruft implizit den Konstruktor der Oberklasse <i>Product</i> auf.		
Bei der Klasse <i>ShoppingCartItem</i> handelt es sich um eine Top-Level-Klasse.		
Drop-Down-Listen werden in Swing mit Hilfe der Klasse <i>JComboBox</i> realisiert.		
Der formale Typparameter <i>T</i> der Klasse <i>ShoppingCart</i> ist kovariant definiert.		
Die Aufzählung <i>ClassOfGoods</i> könnte als geschachtelte Klasse der Klasse <i>Goods</i> implementiert werden.		
Die Klasse <i>Goods</i> könnte zusätzlich zur Schnittstelle <i>Comparable</i> auch die Schnittstelle <i>Comparator</i> implementieren.		
Würde man beim Attribut <i>store</i> der Klasse <i>CornerShop</i> den Datentyp von <i>TreeMap</i> auf <i>HashMap</i> ändern, wäre die Reihenfolge der Elemente eine andere.		
Beim Insertsort wird dem unsortierten Teil das jeweils kleinste Element entnommen und dem sortierten Teil angehängt.		
Die Ausnahme <i>OutOfStockException</i> wird in der Klasse <i>CornerShop</i> behandelt.		
Das sogenannte Diamantenproblem verdankt seinen Namen den beiden spitzen Klammern bei der Festlegung von formalen Typparametern.		
Beim Überschreiben einer Methode muss die Methodensignatur erhalten bleiben, der Datentyp des Rückgabewertes kann jedoch geändert werden.		

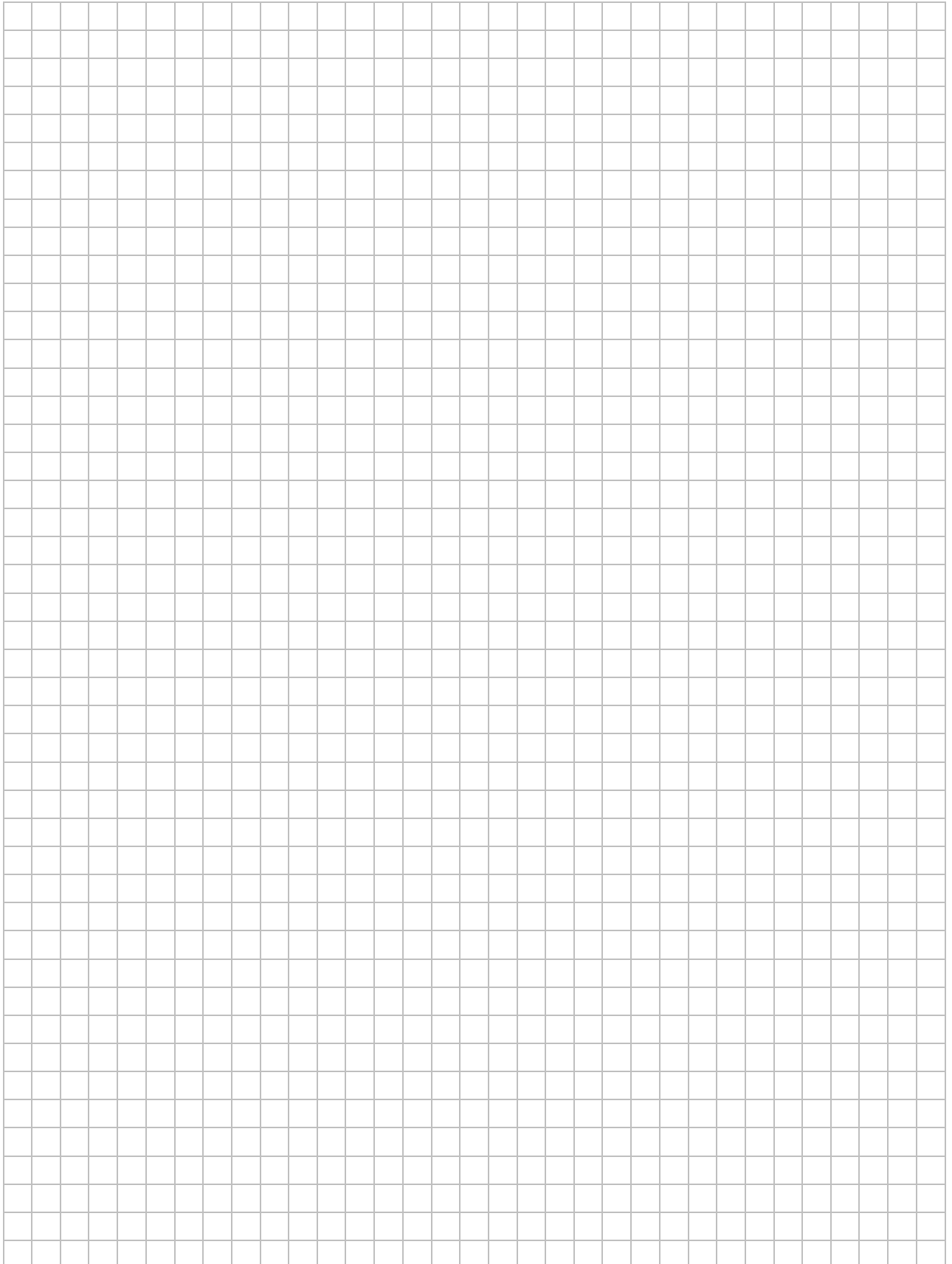
Aufgabe 1.2 (6 Punkte)

Erläutern Sie kurz anhand der Klassen *Product*, *Goods* und *Service*, was man in der Programmierung unter den Begriffen *Polymorphie*, *Upcast* und *Downcast* versteht, welches Risiko ein Downcast mit sich bringt und wie dieses Risiko vermieden werden kann.

A large grid of graph paper, consisting of 20 columns and 30 rows of small squares, intended for the student to write their answer to the question.

Aufgabe 1.3 (2 Punkte)

Erläutern Sie kurz, warum der Konstruktor der Aufzählung *ClassOfGoods* privat definiert ist.

A large grid of graph paper, consisting of 30 columns and 30 rows of small squares, intended for the student to write their answer.

Aufgabe 1.4 (6 Punkte)

Skizzieren Sie das Entwurfsmuster *Model-View-Controller* und erläutern Sie kurz
einen Vorteil dieses Entwurfsmusters.

A large grid of graph paper, consisting of 30 columns and 30 rows of small squares, intended for sketching and writing the answer to the task.

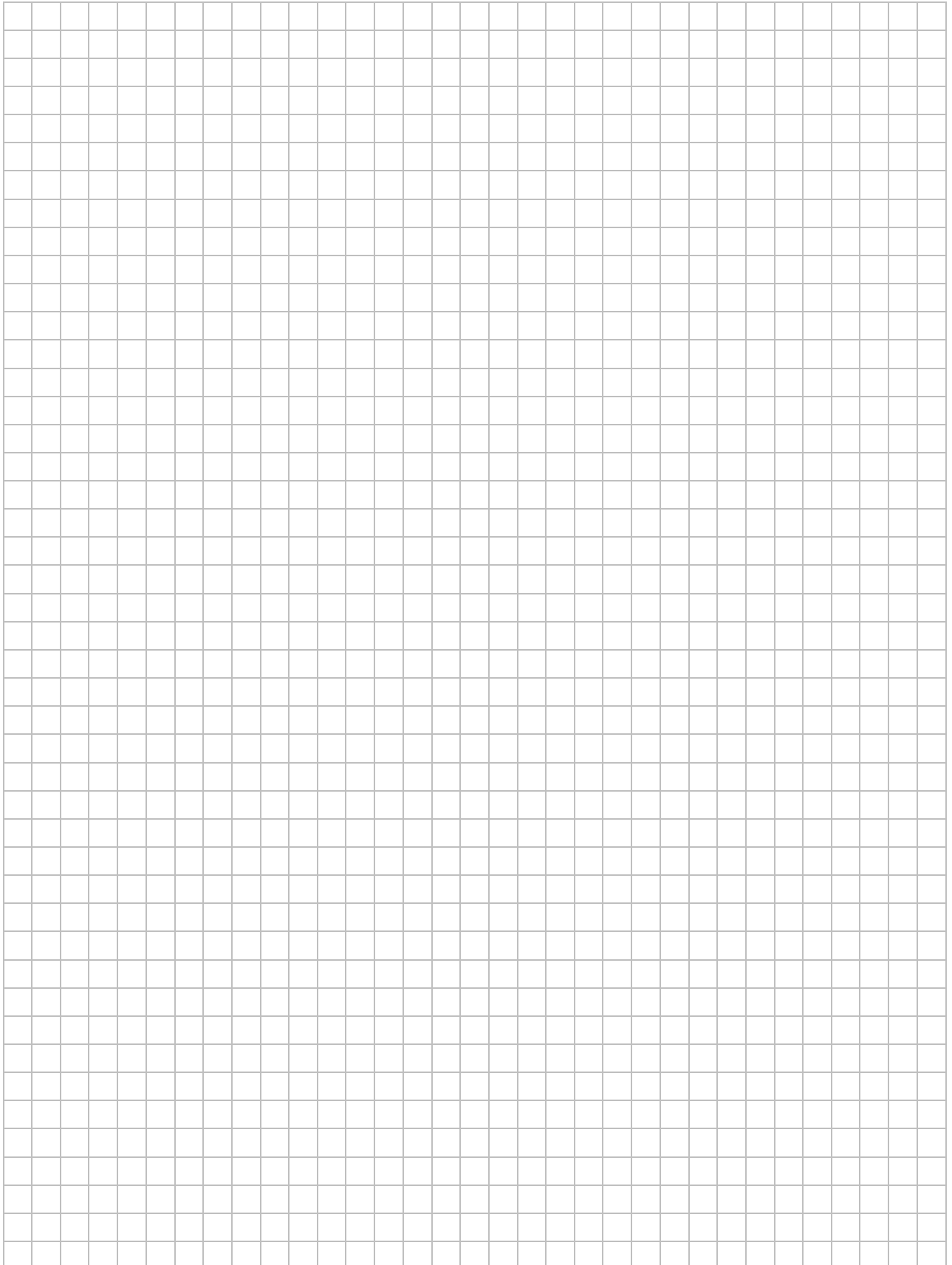
Aufgabe 1.5 (4 Punkte)

Skizzieren Sie die Ereignisbehandlung in Java. Wie wird dieses Modell bezeichnet?

A large grid of graph paper, consisting of 30 columns and 30 rows of small squares, intended for sketching the event handling model in Java.

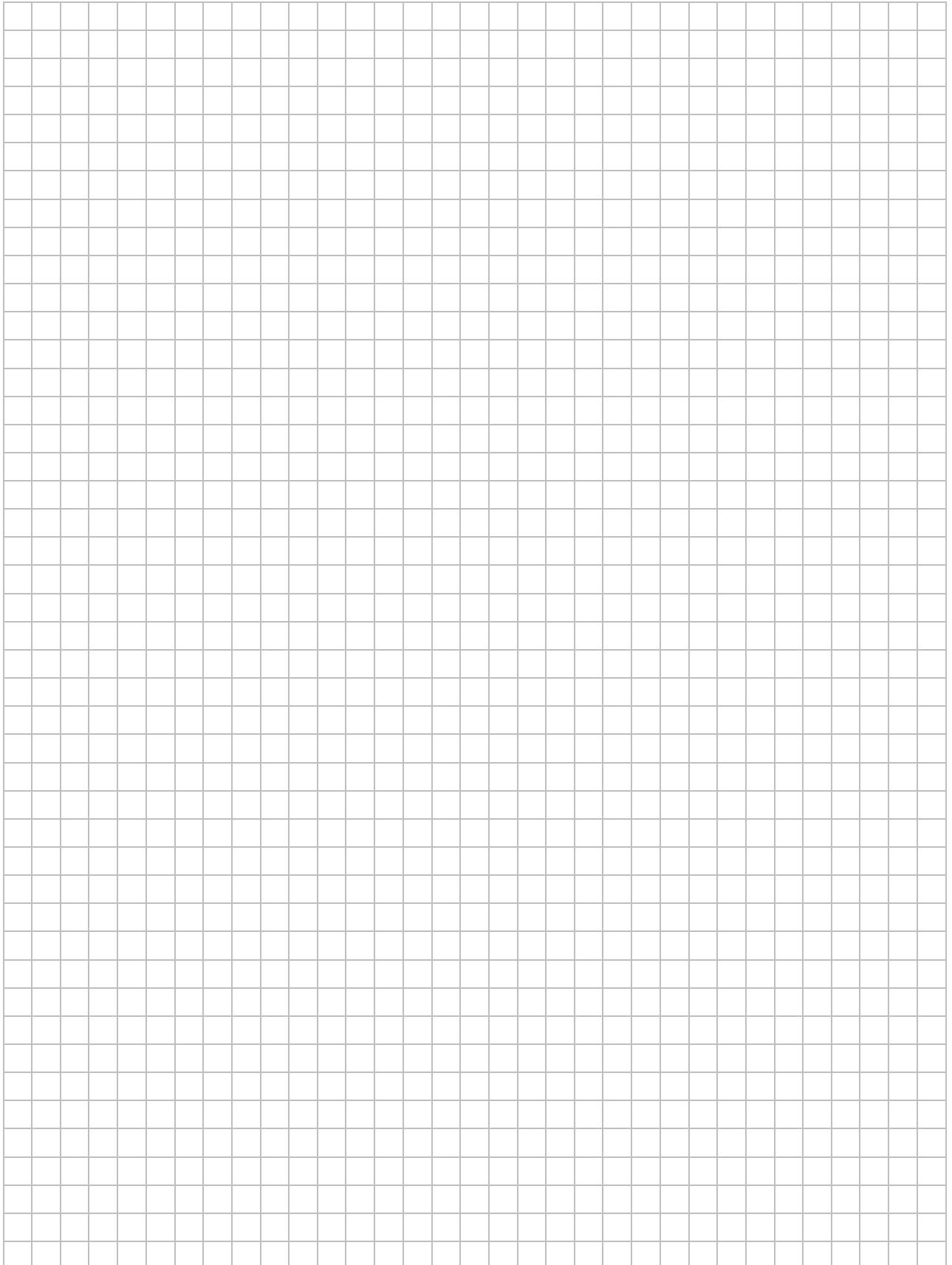
Aufgabe 1.6 (3 Punkte)

Erläutern Sie kurz die Besonderheit der Schnittstelle *Serializable* und erläutern Sie kurz, welchen Zweck die Konstante *serialVersionUID* erfüllt.

A large grid of graph paper, consisting of 30 columns and 40 rows of small squares, intended for the student to write their answer to the question.

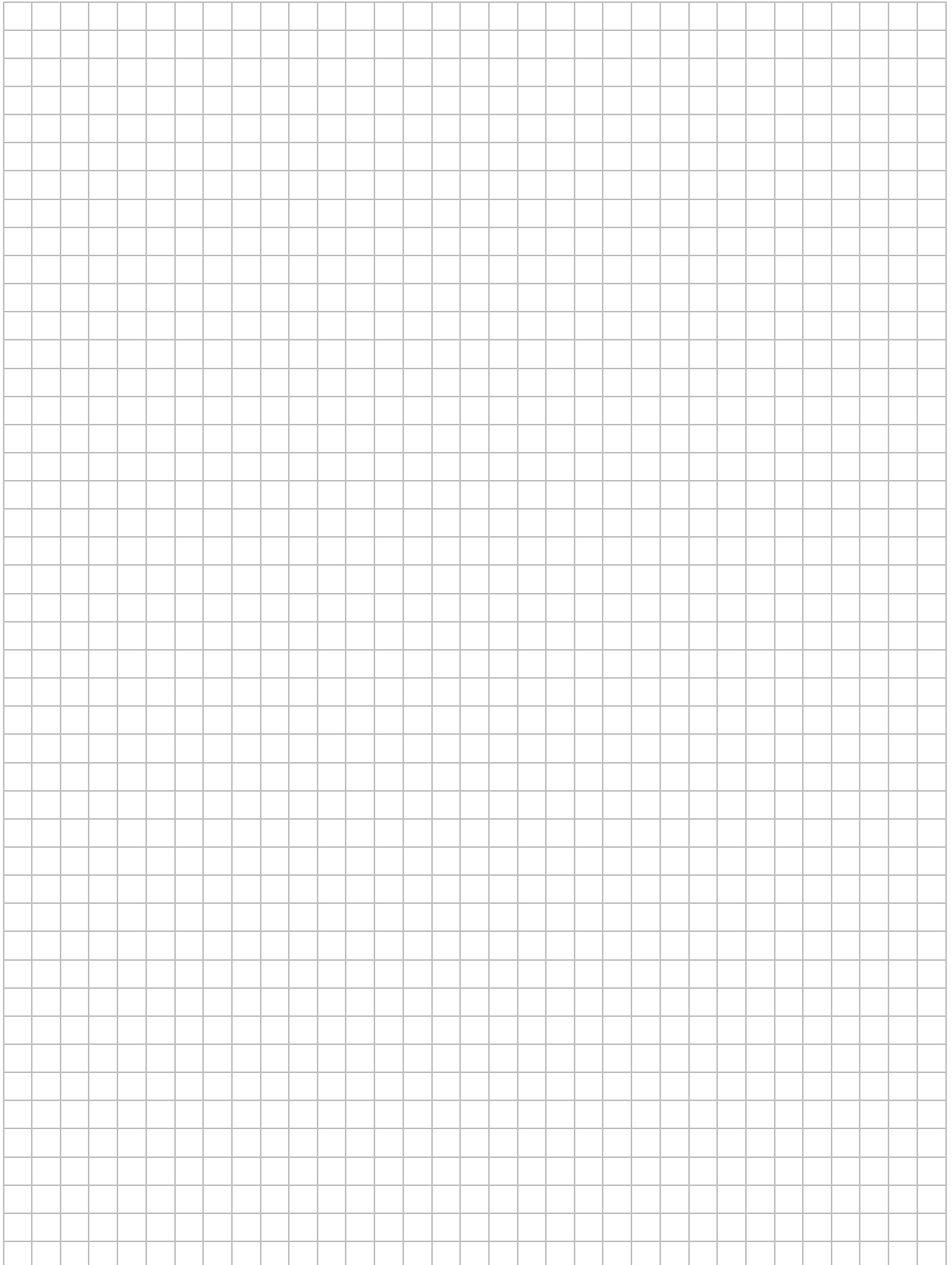
Aufgabe 1.7 (8 Punkte)

Erläutern Sie kurz, wozu der Typ ? in Java verwendet wird und erläutern Sie kurz die drei damit einhergehenden Varianztypen.

A large grid of graph paper, consisting of 30 columns and 30 rows of small squares, intended for the student to write their answer to the question.

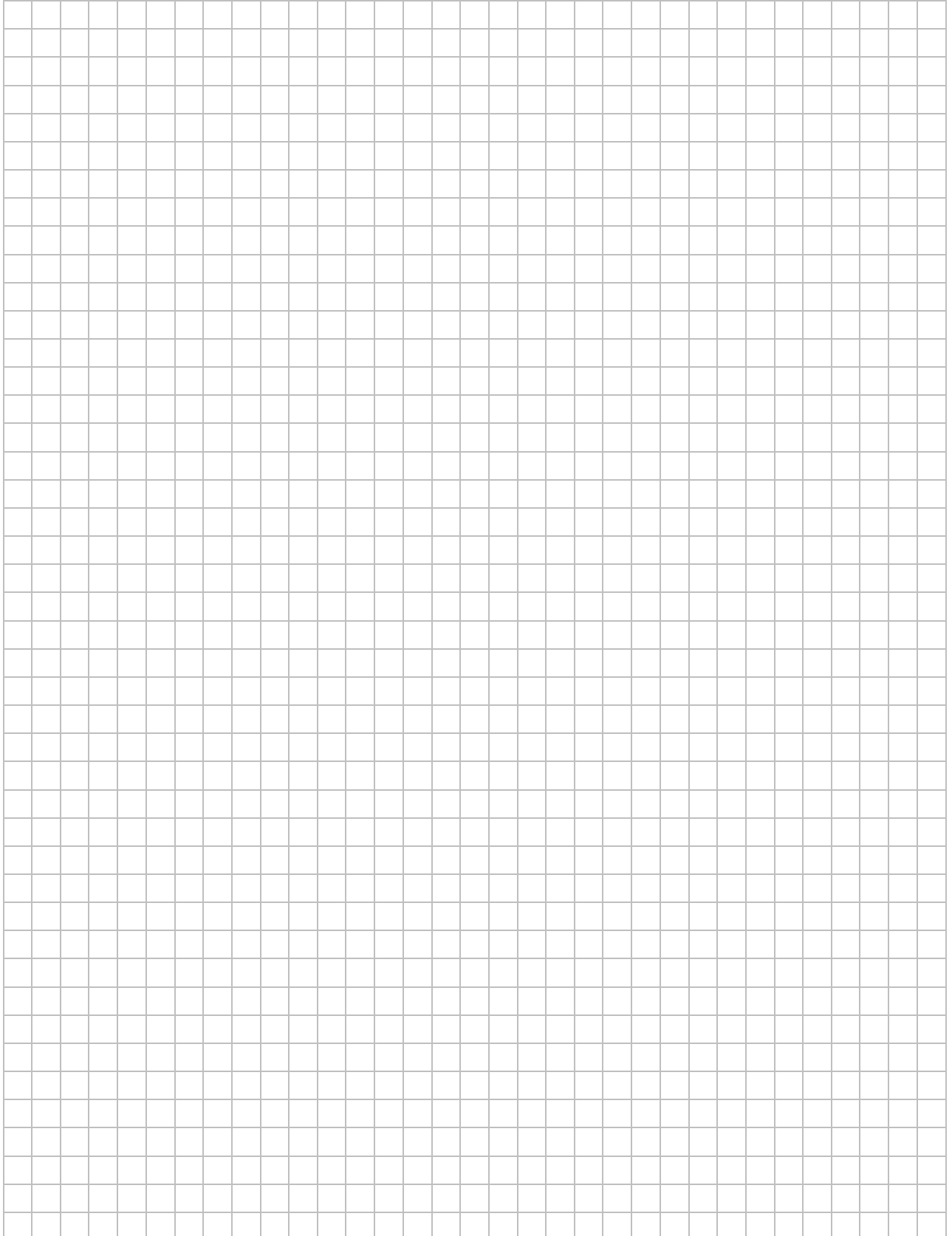
Aufgabe 1.8 (3 Punkte)

Erläutern Sie kurz, was man in Java unter *Collections* versteht und benennen Sie die vier Collection-Grundformen.

A large grid of graph paper, consisting of 30 columns and 30 rows of small squares, intended for the student to write their answer.

Aufgabe 1.9 (4 Punkte)

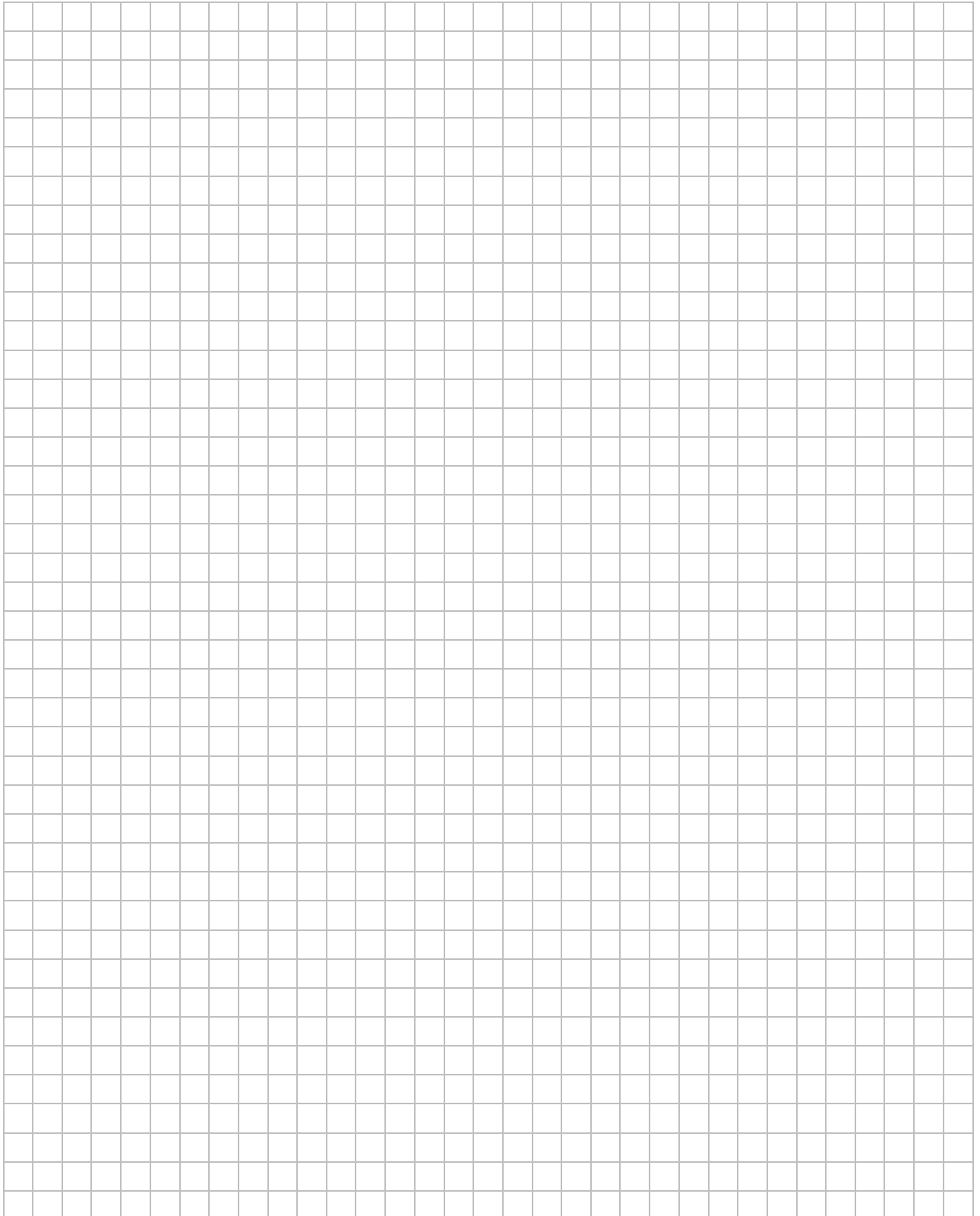
Erläutern Sie kurz den wesentlichen Unterschied zwischen der *Binären Suche* und der *Interpolationssuche* und erläutern Sie kurz, warum es sinnvoll ist, Datensammlungen vor der Suche zu sortieren.

A large grid of graph paper, consisting of 20 columns and 30 rows of small squares, intended for the student to write their answer to the task.

Teil 2: Praxis (68 Punkte)

Aufgabe 2.1 (4 Punkte)

Erstellen Sie die Schnittstelle *Deable* anhand des gegebenen Klassendiagramms.

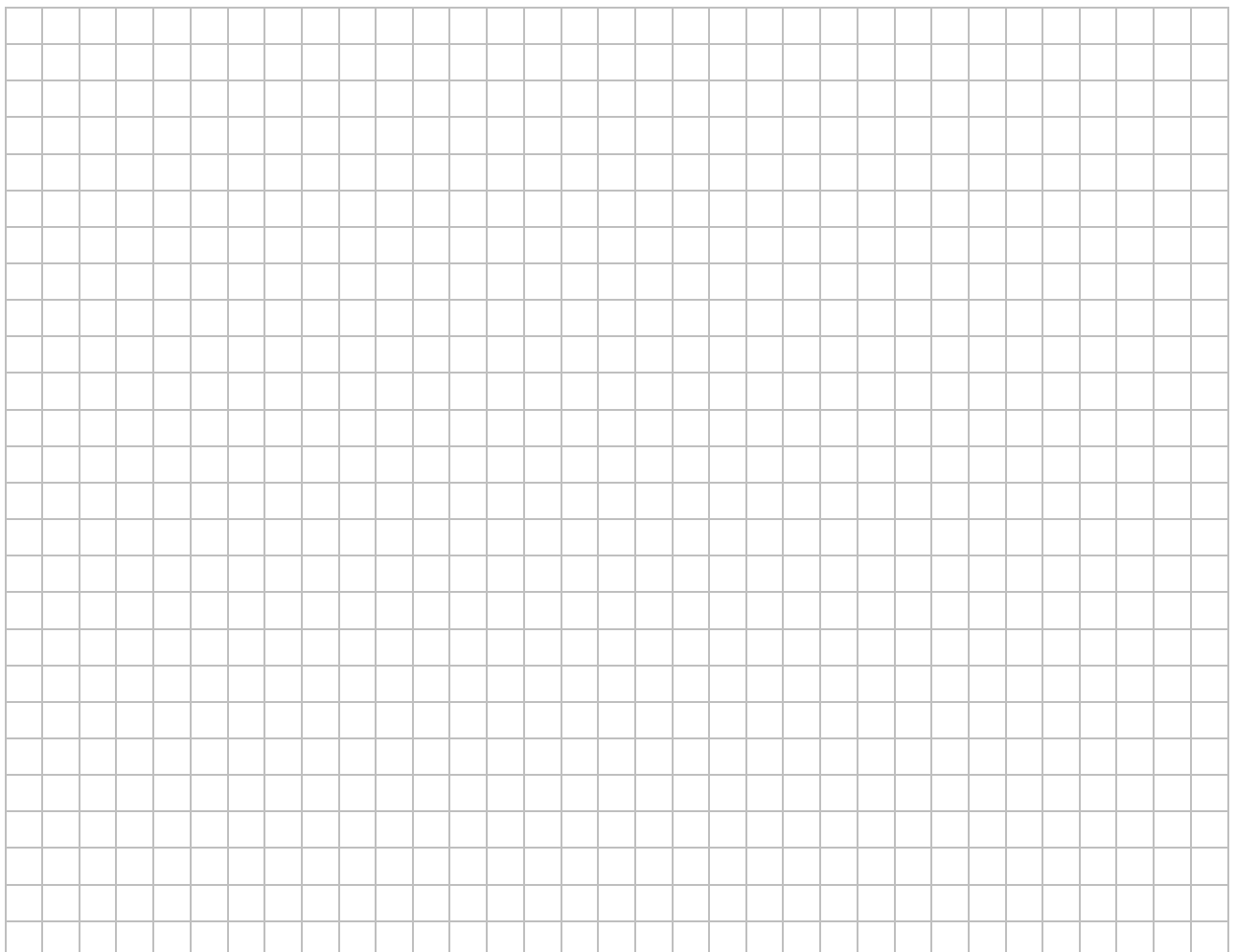


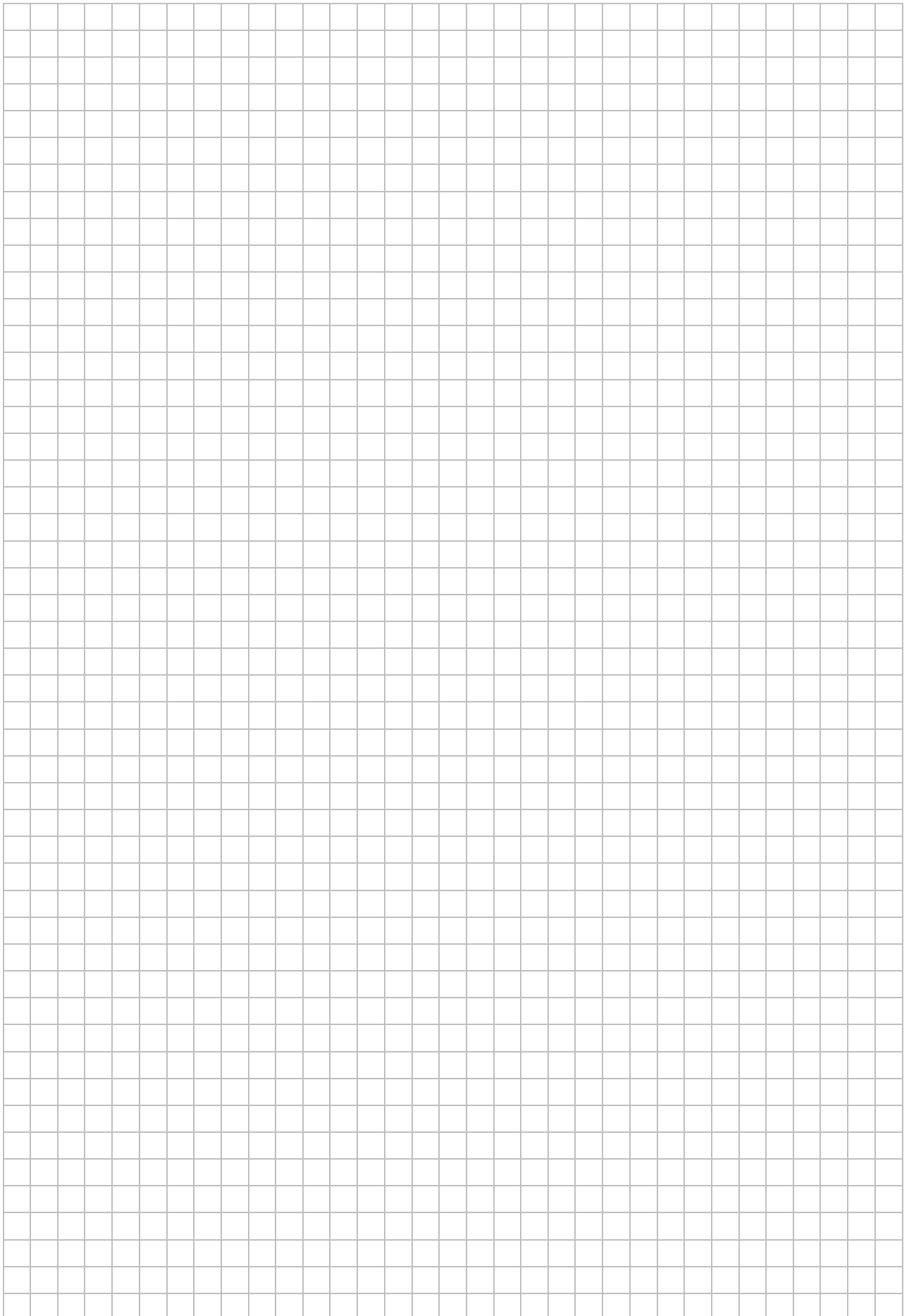
Aufgabe 2.2 (14 Punkte)

Erstellen Sie die Klassen *ShoppingCart* und *ShoppingCartItem* anhand des gegebenen Klassendiagramms.

Hinweise:

- Der Konstruktor der Klasse *ShoppingCart* soll das Attribut *items* initialisieren!
- Die Methode *addItem(T, Integer)* der Klasse *ShoppingCart* soll dem Warenkorb ein Produkt samt Mengenangabe hinzufügen!
- Die Methode *getItems()* der Klasse *ShoppingCart* soll alle Warenkorbeinträge zurückgeben!
- Die Methode *getTotal()* der Klasse *ShoppingCart* soll die Gesamtsumme des Warenkorbs zurückgeben! Die Gesamtsumme berechnet sich aus der Summe aller Zwischensummen!
- Der Konstruktor der Klasse *ShoppingCartItem* soll alle Attribute initialisieren!
- Die Methode *getProduct()* der Klasse *ShoppingCartItem* soll das Produkt zurückgeben!
- Die Methode *getAmount()* der Klasse *ShoppingCartItem* soll die Menge zurückgeben!
- Die Methode *getSubTotal()* der Klasse *ShoppingCartItem* soll die Zwischensumme zurückgeben! Die Zwischensumme berechnet sich aus dem Produkt von Preis und Menge!





Aufgabe 2.3 (4 Punkte)

Implementieren Sie die Methode *compareTo(Goods)* der Klasse *Goods* so, dass Waren absteigend anhand ihrer Warengruppen-Beschreibung sortiert werden!

```
public class Goods extends Product
    implements Comparable<Goods>, Serializable {
...
    public int compareTo(Goods g) {
```

$$\left. \begin{array}{l} \{ \\ \} \end{array} \right\}$$

Aufgabe 2.4 (12 Punkte)

Implementieren Sie die Methoden *getGoodsByDescription(String)* und *addGoodsToShoppingCart(Goods, Integer)* der Klasse *CornerShop*.

Hinweise:

- Die Methode *getGoodsByDescription(String)* soll zur eingegebenen Waren-Bezeichnung die entsprechende Ware im Lager suchen und samt verfügbarer Menge zurückgeben!
- Achten Sie bei der Verwendung von generischen Klassen (wie z.B. bei der Klasse *Iterator*) darauf, dass die Typsicherheit gewährleistet ist!
- Die Methode *addGoodsToShoppingCart(Goods, Integer)* soll dem Warenkorb die eingegebene Ware samt eingegebener gewünschter Menge hinzufügen! Überschreitet die gewünschte Menge die verfügbare Menge im Lager, soll die Ausnahme *OutOfStockException* ausgelöst werden

```
public class CornerShop implements Dealable<Goods> {  
    private String name;  
    private TreeMap<Goods, Integer> store;  
    private ShoppingCart<Goods> shoppingCart;  
  
    public CornerShop(String name) {  
        this.name = name;  
        store = new TreeMap<>();  
        clearShoppingCart();  
    }  
}
```

...

Aufgabe 2.5 (8 Punkte)

Implementieren Sie die Methode *loadStore()* der Klasse *CornerShop*.

Hinweise:

- Die Methode *loadStore()* soll alle Waren samt verfügbarer Menge aus der Datei „C:\store.dat“ auslesen!
- Die Datei „C:\store.dat“ beinhaltet genau ein Objekt vom Datentyp *TreeMap<Goods, Integer>*!
- Fangen Sie die Ausnahmen *ClassNotFoundException* und *IOException* ab und lösen Sie als Behandlungsmaßnahme die Ausnahme *StoreIOException* aus!

```
public class CornerShop implements Dealable<Goods> {  
    private String name;  
    private TreeMap<Goods, Integer> store;  
    private ShoppingCart<Goods> shoppingCart;  
  
    public CornerShop(String name) {  
        this.name = name;  
        store = new TreeMap<>();  
        clearShoppingCart();  
    }  
    ...  
    public void loadStore() throws StoreIOException{
```


Aufgabe 2.6 (12 Punkte)

Erweitern Sie den Konstruktor der Klasse *CornerShopFrame* so, dass die gegebene grafische Benutzeroberfläche erzeugt wird. Fügen Sie zudem für die beiden Drucktasten *Ware zum Warenkorb hinzufügen* und *Kauf abschließen* Ereignisbehandler für Ereignisse vom Typ *ActionEvent* hinzu.

Hinweise:

- Verwenden Sie als Ereignisbehandler die Klasse *CornerShopController*!
- Die Methode *setColumnIdentifiers(Object[])* der Klasse *DefaultTableModel* legt die Spaltenbezeichnung für die verwaltete Tabelle fest!
- Die statische Konstante *Y_AXIS* der Klasse *BorderLayout* legt eine vertikale Element-Ausrichtung fest, die Konstante *X_AXIS* eine horizontale!
- Die Klasse *BorderLayout* stellt zur Anordnung der Elemente die Konstanten *CENTER*, *LINE_END*, *LINE_START*, *PAGE_END* und *PAGE_START* bereit!

```
public CornerShopFrame() {  
  
    /*  
     * 1. Waren-Label-Container, Waren-Tabelle (ohne Inhalt),  
     *    Waren-Container und den "linken Container" definieren  
     */  
    goodsLabel = new JLabel("Waren");  
    goodsLabelContainer = new JPanel();  
    goodsTableModel = new DefaultTableModel();  
    goods = new JTable(goodsTableModel);  
    goodsContainer = new JScrollPane(goods);  
    leftContainer = new JPanel();  
}
```

```
/*  
 * 2. Warenkorb-Label-Container, Warenkorb-Tabelle (ohne Inhalt),  
 *   Warenkorb-Container, Gesamtsummen-Container und  
 *   den "rechten Container" definieren  
 */  
shoppingCartLabel = new JLabel("Warenkorb");  
shoppingCartLabelContainer = new JPanel();  
shoppingCartTableModel = new DefaultTableModel();  
shoppingCart = new JTable(shoppingCartTableModel);  
shoppingCartContainer = new JScrollPane(shoppingCart);  
totalLabel = new JLabel("Gesamtsumme:");  
total = new JLabel();  
totalContainer = new JPanel();  
rightContainer = new JPanel();
```

```
/*  
 * 3. Funktions-Container definieren  
 */  
functionsContainer = new JPanel();  
addGoodsToShoppingCart =  
    new JButton("Ware zum Warenkorb hinzufügen");  
finishPurchase = new JButton("Kauf abschließen");
```

```
/*  
 * 4. Haupt-Container definieren  
 */  
mainContainer = new JPanel();
```

```
/*  
 * 5. Ereignisbehandler hinzufügen  
 */
```

```
setSize(1000, 300);  
setLocationRelativeTo(null);  
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
add(mainContainer);
```

```
}
```

Aufgabe 2.7 (14 Punkte)

Wenden Sie das Quicksort-Verfahren auf die nachfolgende Liste an. Schreiben Sie hierzu für jeden Durchlauf die jeweilige Reihenfolge der Zahlen auf, kreisen Sie den jeweiligen Teiler ein und markieren sie die entstandenen neuen Bereiche!

Hinweis: Verwenden Sie die gegebene Implementierung des Quicksort!

Index	0	1	2	3	4	5	6	7	8
0	8								
1	2								
2	6								
3	1								
4	7								
5	5								
6	4								
7	9								
8	3								

Durchgang	l	r	c	i	j	l/j	i/r
1							
2							
3							
4							
5							
6							
7							
8							

