**Aufgabe: Einkaufsplattform für Produkte**

Du sollst eine Anwendung für eine Einkaufsplattform erstellen, auf der verschiedene Arten von Produkten verkauft werden. Es gibt eine Basisklasse für allgemeine Produkte, von der dann spezielle Produktarten wie Elektronik und Lebensmittel erben. Dabei spielen Vererbung und Exception Handling eine wichtige Rolle.

**Anforderungen**

1. **Vererbung**
   * Erstelle eine Basisklasse Produkt, die allgemeine Eigenschaften wie name, preis (als double), und lagerbestand (als int) enthält.
   * Erstelle zwei Unterklassen: Elektronik und Lebensmittel. Diese erben von der Basisklasse Produkt, haben aber zusätzliche spezifische Attribute:
     + Elektronik-Produkte haben ein garantieJahre (als int).
     + Lebensmittel-Produkte haben ein haltbarkeit (als String, z.B. "2025-12-31").
2. **Ausnahmebehandlung (Exceptions)**
   * Wenn ein Kunde ein Produkt kauft, soll überprüft werden, ob genug Lagerbestand vorhanden ist. Falls nicht, soll eine benutzerdefinierte Ausnahme NichtGenugLagerbestandException geworfen werden.
   * Falls der Preis eines Produkts negativ ist, soll eine Ausnahme InvalidPreisException geworfen werden.
   * Wenn das Haltbarkeitsdatum eines Lebensmittelprodukts ungültig ist (z.B. in der Vergangenheit), soll eine Ausnahme InvalidHaltbarkeitsdatumException geworfen werden.

**Details der Klassenstruktur**

1. **Basisklasse Produkt**
   * Attribute:
     + name (String)
     + preis (double)
     + lagerbestand (int)
   * Methoden:
     + Getter und Setter für alle Attribute.
     + Methode verkaufen(int menge), die überprüft, ob genügend Lagerbestand vorhanden ist. Wenn ja, wird der Lagerbestand reduziert.
2. **Unterklasse Elektronik**
   * Erbt von Produkt
   * Zusätzliche Attribute:
     + garantieJahre (int)
   * Methoden:
     + Ein Konstruktor, der alle Felder initialisiert und eine Methode, die die Garantie anzeigt.
3. **Unterklasse Lebensmittel**
   * Erbt von Produkt
   * Zusätzliche Attribute:
     + haltbarkeit (String, z.B. "2025-12-31")
   * Methoden:
     + Ein Konstruktor, der alle Felder initialisiert und eine Methode, die das Haltbarkeitsdatum überprüft.
4. **Benutzerdefinierte Ausnahmen**
   * NichtGenugLagerbestandException: Wird geworfen, wenn nicht genügend Exemplare eines Produkts verfügbar sind.
   * InvalidPreisException: Wird geworfen, wenn der Preis negativ ist.
   * InvalidHaltbarkeitsdatumException: Wird geworfen, wenn das Haltbarkeitsdatum eines Lebensmittels ungültig ist.
5. **Hauptklasse Einkaufsplattform**
   * Diese Klasse verwaltet die Produktliste und führt Transaktionen durch. Sie muss den Lagerbestand von Produkten verwalten und die entsprechenden Ausnahmen behandeln.

**Aufgabe 1: Rabatt-System für Produkte mit abstrakter Klasse**

Erstelle eine zusätzliche Funktion, die es ermöglicht, für jedes Produkt einen Rabatt anzuwenden. Der Rabatt wird durch einen Prozentsatz angegeben (z.B. 10% Rabatt auf den Preis). Die Berechnung des neuen Preises muss sicherstellen, dass der Rabatt korrekt angewendet wird und der Preis nicht negativ wird. Verwende dabei eine abstrakte Methode in einer erweiterten abstrakten Klasse.

Anforderungen:

Erstelle eine abstrakte Methode anwendenRabatt(double prozent) in der Basisklasse Produkt, die in den Unterklassen überschrieben wird.

In der Methode anwendenRabatt soll geprüft werden, ob der rabattierte Preis negativ wird. Falls ja, soll eine Ausnahme InvalidPreisException geworfen werden.

Die Methode anwendenRabatt soll von den Unterklassen Elektronik und Lebensmittel überschrieben werden und unterschiedliche Rabattanwendungen haben:

Elektronik: Es soll ein fester Rabatt von 5% gewährt werden.

Lebensmittel: Lebensmittel erhalten einen Rabatt basierend auf dem Prozentsatz, der als Parameter übergeben wird.

**Aufgabe 2: Bestandsübersicht und Warnsystem mit Casting**

Erstelle eine Methode bestandsuebersicht() in der Klasse Einkaufsplattform, die alle Produkte im System auflistet und anzeigt, ob der Lagerbestand unter 10 Stück liegt. Produkte, bei denen der Lagerbestand unter 10 liegt, sollen mit einer Warnung markiert werden. Implementiere zudem Casting, um zwischen den Produktarten zu unterscheiden (z.B. Elektronik und Lebensmittel).

Anforderungen:

1. Erstelle eine Methode bestandsuebersicht(), die alle Produkte in einer Liste iteriert und die Produktnamen und den Lagerbestand anzeigt.
2. Wenn der Lagerbestand unter 10 liegt, soll eine Warnung ausgegeben werden: Warnung: Niedriger Lagerbestand für [Produktname].
3. Führe ein Casting durch, um zusätzliche Informationen für spezifische Produkttypen anzuzeigen:
   * Bei Elektronik-Produkten soll die Garantie angezeigt werden.
   * Bei Lebensmittel-Produkten soll das Haltbarkeitsdatum angezeigt werden.

**Aufgabe: Unterschied zwischen checked und unchecked Exceptions**

**Aufgabenstellung:**

Du sollst eine kleine Anwendung erstellen, die mit verschiedenen Arten von Ausnahmen arbeitet, um den Unterschied zwischen **checked** und **unchecked** Exceptions zu demonstrieren.

**1. Erstelle die Klasse Mathematik**  
In dieser Klasse soll eine Methode teilen(int a, int b) erstellt werden, die zwei Zahlen durchführt. Wenn der Divisor b null ist, soll eine **checked Exception** geworfen werden, z.B. DivisionDurchNullException.

**2. Erstelle die Klasse Benutzer**  
In dieser Klasse soll eine Methode alterPruefen(int alter) erstellt werden, die überprüft, ob das Alter negativ ist. Wenn das Alter negativ ist, soll eine **unchecked Exception** geworfen werden, z.B. InvalidAgeException (die eine RuntimeException ist).

**3. Verwende beide Methoden in der Main-Klasse.**  
In der main-Methode rufst du die Methoden teilen() und alterPruefen() auf. Stelle sicher, dass du:

* Die **checked Exception** korrekt mit einem try-catch-Block behandelst.
* Die **unchecked Exception** entweder explizit fängst oder nicht fängst, um den Unterschied zu zeigen (optional kannst du auch zeigen, wie sie durch das Fehlen eines try-catch-Blocks eine Anwendung zum Absturz bringen kann).

**Anforderungen:**

1. **Erstelle eine DivisionDurchNullException**:  
   Dies ist eine benutzerdefinierte **checked Exception**. Diese Exception wird geworfen, wenn versucht wird, eine Zahl durch null zu teilen.
2. **Erstelle eine InvalidAgeException**:  
   Dies ist eine benutzerdefinierte **unchecked Exception** (von RuntimeException abgeleitet). Diese Exception wird geworfen, wenn ein ungültiges (negatives) Alter eingegeben wird.
3. **Verwende die Exceptions in der main-Methode**:
   * Zeige den Unterschied, indem du versuchst, eine Division durch null durchzuführen und die DivisionDurchNullException mit einem try-catch-Block abfängst.
   * Zeige ebenfalls, wie die InvalidAgeException geworfen wird und entweder abgefangen oder nicht abgefangen wird.

**Hinweise:**

* **Checked Exception:** Eine **checked Exception** (wie DivisionDurchNullException) muss entweder in der Methode, die sie wirft, mit throws deklariert werden, oder sie muss in einem try-catch-Block abgefangen werden. Der Compiler verlangt, dass mit diesen Ausnahmen explizit umgegangen wird.
* **Unchecked Exception:** Eine **unchecked Exception** (wie InvalidAgeException) ist eine Ausnahme, die nicht zwingend abgefangen werden muss. Sie ist eine Unterklasse von RuntimeException, und der Compiler erzwingt nicht, dass mit ihr umgegangen wird.

**Erweiterung (Optional):**

* Füge einen weiteren Block hinzu, der den Versuch zeigt, beide Exceptions zu behandeln oder die Anwendung bei der unchecked Exception abstürzen zu lassen, um den Unterschied noch klarer zu machen.