Wintersemester 2021 / 22		Seite	1 von 15
Fachbereich:	Informationstechnik	Studiengang:	TIB, SWB, IEP
Prüfungsfach:	OOS1	Prüfungsnummer.:	1052027
Hilfsmittel:	handschriftliche Notizen (auch gedruckt) 2 Blätter DIN A4 beidseitig	Zeit:	90 min
Nachname: Vorname:		Matrikelnummer:	

Hinweis: Der auf den Blättern jeweils freigelassene Raum reicht im Allgemeinen vollständig für die stichwortartige Beantwortung der Fragen, bzw. für die Lösungen aus. Tragen Sie daher auf jedem Blatt Ihren Namen und Ihre Matrikelnummer ein und nutzen Sie diese Blätter zur Abgabe Ihrer Antworten und Lösungen.

Aufgabe 1: Allgemeine Fragen (ca. 20 Min.)

Beurteilen Sie die folgenden allgemeinen Aussagen. Machen Sie jeweils ein Kreuzchen in der Spalte “wahr” oder “falsch”. Begründen Sie jeweils Ihre Wahl.

Aussage	wahr	falsch
Der Wert von konstanten Attributen ist in jedem Objekt gleich. Begründung:		
In einer abstrakten Klasse dürfen nur abstrakte Methoden vorkommen. Begründung Es muss min eine abstrakte Klasse vorkommen		X
Im Destruktor einer abgeleiteten Klasse wird als erstes der Destruktor der Basisklasse aufgerufen. Begründung:		
Der Zugriff auf ein Zeichen eines Strings mittels der Methode `at` ist nicht so effizient wie mit dem Operator [ ], dafür aber sicherer. Begründung:		

Von abstrakten Klassen können keine Objekte erzeugt werden. Begründung:		
static-Methoden einer Klasse können nur aufgerufen werden, wenn es mindestens ein Objekt der Klasse gibt. Begründung:		
Von einer Aggregation spricht man dann, wenn das aggregierende (Gesamt-)Objekt die Lebensdauer der aggregierten (Teil-)Objekte kontrolliert. Begründung:		
Mit `class A { int a; friend B; };` kann die Klasse A auf die privaten Elemente der Klasse B zugreifen. Begründung:		
Einem Kopierkonstruktor kann ein Objekt entweder call-by-value oder call-by-reference übergeben werden Begründung:		
In einer abgeleiteten Klasse kann man eine Methode der Basisklasse überladen. Begründung:		

Aufgabe 2: Klassen, Attribute, Methoden, Operatoren (17 min)

Die Klasse `Ware` ist die Basisklasse aller Waren, die in einem Warenverwaltungssystem verwaltet werden können (vgl. dazu auch Aufgaben 3 und 4). Sie soll die folgenden Anforderungen realisieren:

a) Eine Ware wird beschrieben durch das Attribut `name`, das den Namen der Ware (vom Typ `string`) speichert. Dieses Attribut ist nicht öffentlich zugänglich. Das Attribut `name` wird durch einen parametrisierten Konstruktor gesetzt und kann danach nicht mehr geändert werden.
b) Die Klasse hat einen Operator == als Methode. Dieser überprüft, ob die Namen der beiden Objekte gleich sind.
c) Die Klasse hat eine Methode `toString()`, die zu einer Ware den Namen zurückgibt.

Die Klasse `Datum` soll die folgenden Anforderungen realisieren:

a) Ein Datum wird beschrieben durch die Attribute `tag`, `monat` und `jahr`, jeweils vom Typ `int`. Die Attribute sind nicht öffentlich zugänglich. Sie werden durch einen parametrisierten Konstruktor gesetzt und können danach nicht mehr geändert werden.
b) Die Klasse hat einen Operator < als Methode. Dieser überprüft, ob das Datum vor dem in der Methode übergebenen Datum liegt.
c) Die Klasse hat eine Methode `toString()`, die das Datum als String in der Form <tag>.<monat>.<jahr> (z.B. 2.2.2022) zurückgibt.
d) Die Klasse hat eine statische Methode `heute()`, die das heutige Datum zurückgibt.

Ergänzen Sie die folgenden Programmgerippe.
Trennen sie die Umsetzung sinnvoll in Headerdateien und Implementierungsdateien.
Die Deklaration der Klassen Ware und Datum ist bereits vorgegeben. Includes müssen ggf. noch angegeben werden.

`Ware.h`

```
// alles was für die Klassendeklaration benötigt wird
```

Klassendeklaration der Klasse `Ware`

```
class Ware {
// Instanzvariablen
    const string name;
// Konstruktor
public:
    Ware(string);
// Methoden
    virtual string toString() const;
    bool operator==(const Ware&) const;
};
```

Ware.cpp

```
// alles was für die Klassendefinition benötigt wird
```

Klassendefinition der Klasse Ware

```
// Konstruktor




// Methoden
```

Datum.h

```
// alles was für die Klassendeklaration benötigt wird
```

Klassendeklaration der Klasse Datum

```
class Datum
{
// Instanzvariablen
    const int tag, monat, jahr;
// Konstruktor
public:
    Datum(int tag, int monat, int jahr);
// Methoden
    string toString() const;
    bool operator<(Datum) const;
    static Datum heute();
};
```

Datum.cpp

```
// alles was für die Klassendefinition benötigt wird
```

Klassendefinition der Klasse `Datum`

```
// Konstruktor




// Methoden
```

Aufgabe 3: Vererbung (14 min)

Leiten Sie die Klasse `Lebensmittel` von der Klasse `Ware` ab mit den folgenden Eigenschaften.
a) Die Klasse `Lebensmittel` hat ein zusätzliches Attribut `mhd` vom Typ `Datum`, dem Mindesthaltbarkeitsdatum des Produktes.
b) Ein Konstruktor der Klasse erhält den Namen und das Mindesthaltbarkeitsdatum als Parameter.
c) Die Methode `getMHD()` gibt das Mindesthaltbarkeitsdatum zurück.
d) Eine `toString()`-Methode überschreibt die Methode der Basisklasse. Sie gibt einen String im Format <name>, MHD: <tag>.<monat>.<jahr> zurück, z.B. "Joghurt, MHD: 1.12.2021".

Ergänzen Sie die folgenden Programmgerippe.
Trennen sie die Umsetzung sinnvoll in Headerdateien und Implementierungsdateien.
`Lebensmittel.h`

```
// alles was für die Klassendeklaration benötigt wird
```

Klassendeklaration der Klasse `Lebensmittel`

```
class Lebensmittel                    {
// Instanzvariablen




// Konstruktor




// Methoden









};
```

Lebensmittel.cpp

```
// alles was für die Klassendefinition benötigt wird
```

Klassendefinition der Klasse `Lebensmittel`

```
// Konstruktor




// Methoden
```

Aufgabe 4: Klassen, Operatoren, Vektoren (13 min)

Die Klasse Warenlager enthält einen Speicher für Waren. Dadurch, dass die Klasse Ware beschrieben ist, können Sie die Klasse Warenlager auch realisieren, ohne die Aufgabe 2 lösen zu müssen.
Die Klasse Warenlager soll die im Folgenden genannten Anforderungen realisieren:

a) Die Klasse enthält als Attribute ein Vektor aus Pointern auf `Ware` (Attribut `lager`) und die Größe des Warenlagers (Attribut `kapazitaet`).
b) Ein Konstruktor erhält über einen Parameter die Größe des Warenlagers und setzt diese entsprechend.
c) Die Klasse hat eine Methode `lagere()` mit einem Pointer auf `Ware` als Parameter. Die Methode fügt den Pointer auf die `Ware` dem Vektor hinzu. Sie gibt die Meldung „`Lager voll`“ auf der Konsole aus, wenn kein Platz mehr im Lager frei ist.
d) Eine weitere Methode `entferne()` mit einem Pointer auf Ware als Parameter sucht die Ware im Lager mit Hilfe des Vergleichsoperators ==. Falls diese Ware im Vektor gespeichert ist, wird der Eintrag entfernt. Dadurch ist die Ware nicht mehr im Lager vorhanden. Diese Methode muss **NICHT** implementiert werden.
e) Die Methode `listeBestand()` druckt die aktuell im Lager gespeicherten Waren aus. Für jedes Element des Vektors, ruft sie die Methode `tostring()` der Ware auf. In der Beschreibung von Aufgabe 5 finden Sie ein Beispiel für die Ausgabe dieser Methode.
f) Die Klasse hat einen Operator `[]` als Methode. Der zweite Operand ist eine ganze Zahl und wird als Index benutzt. Der Operator gibt als Ergebnis den Pointer zurück, der unter dem Index im Vektor `lager` gespeichert ist.
g) Die Methode `belegteLagerplaetze()` gibt die Anzahl der belegten Lagerplätze zurück.

Ergänzen Sie die folgenden Programmgerippe.
Trennen sie die Umsetzung sinnvoll in Headerdateien und Implementierungsdateien.
Die Deklaration der Klasse Warenlager ist bereits vorgegeben. Includes müssen ggf. noch angegeben werden.

Warenlager.h

```
// alles was für die Klassendeklaration benötigt wird
```

Klassendeklaration der Klasse `Warenlager`

```
class Warenlager {
// Instanzvariablen
    int kapazitaet;
    vector<Ware*> lager;
// Konstruktor
public:
    Warenlager(int);
// Methoden
    Warenlager& lagere(Ware*);
    Warenlager& entferne(Ware*); // muss NICHT implementiert werden
    void listeBestand() const;
    Ware* operator[](int) const;
    int belegteLagerplaetze();
};
```

Warenlager.cpp

```
// alles was für die Klassendefinition benötigt wird, falls nötig
```

Klassendefinition der Klasse `Warenlager`

```
// Konstruktor



// Methoden

// Fortsetzung auf Seite 10
```

```
// Fortsetzung von Seite 9
```

Aufgabe 5: Polymorphie und Casts (8 min)

Das unten angegebene Hauptprogramm zeigt die Verwendung der Klassen `Ware`, `Warenlager`, `Datum` und `Lebensmittel`. Die Ausgabe des Programms ist wie folgt:

```
Lager mit 100 Plaetzen:
Spielzeugauto
Joghurt, MHD: 1.12.2021
Fussball
Teller
Milch, MHD: 1.1.2022
Butter, MHD: 31.5.2022
```

In dem Hauptprogramm wird eine Funkion `entferneAbgelaufeneWaren()` aufgerufen. Die Funktion `entferneAbgelaufeneWaren()` untersucht jedes Element des Warenlagers wie folgt:

a) Es wird geprüft, ob das Element auf ein Objekt der Klasse `Lebensmittel` zeigt. Wenn dies der Fall ist, wird das Mindesthaltbarkeitsdatum des Produktes geprüft. Ist dies kleiner/älter (Operator <) als das heutige Datum, muss das Produkt aus dem Lager entfernt werden.
b) Handelt es sich um eine normale Ware (kein Lebensmittel), dann ist nichts weiter zu tun.

Da die Schnittstellen der Klassen `Warenlager` und `Datum` gegeben sind, kann diese Aufgabe unabhängig von der Lösung der Aufgaben 2 und 4 bearbeitet werden.
Ergänzen Sie das folgenden Programmgerippe. Die `main`-Methode ist bereits vorgegeben. Includes müssen ggf. noch angegeben werden.

main.cpp

```
// alles was für das Programm benötigt wird
```

```
void entferneAbgelaufeneWaren(Warenlager &lager);

int main() {
    Warenlager regal(100);
    regal.lagere(new Ware("Spielzeugauto"));
    regal.lagere(new Lebensmittel("Joghurt", Datum(1, 12, 2021)));
    regal.lagere(new Ware("Fussball"));
    regal.lagere(new Ware("Teller"));
    regal.lagere(new Lebensmittel("Milch", Datum(1, 1, 2022)));
    regal.lagere(new Lebensmittel("Butter", Datum(31, 5, 2022)));
    regal.listeBestand();
    entferneAbgelaufeneWaren(regal);
    return 0;
}
void entferneAbgelaufeneWaren(Warenlager &lager) {









}
```

Aufgabe 6: Dynamisches Binden (10 min)

Analysieren Sie die nachfolgende Funktion `main` und schreiben Sie die Ausgabe der Funktion direkt hinter die Zeile.

```
#include <iostream>
using namespace std;

class Tier
{
public:
    virtual void laufe() { cout << "Tier::laeuft" << endl; }
    void fliege() { cout << "Tier::kann nicht fliegen" << endl; }
    void friss() { cout << "Tier::frisst Nahrung" << endl; }
};
class Vogel : public Tier
{
public:
    void laufe() override { cout << "Vogel::laeuft" << endl; }
    virtual void fliege() { cout << "Vogel::fliegt" << endl; }
    void friss() { cout << "Vogel::frisst Wuermer" << endl; }
};
class Papagei : public Vogel
{
public:
    void laufe() override { cout << "Papagei::laeuft" << endl; }
    void fliege() override { cout << "Papagei::fliegt" << endl; }
    virtual void friss() { cout << "Papagei::frisst Koerner" << endl; }
};
int main()
{
    Papagei papagei;
    Vogel vogel;
    Tier* p_tier = &vogel;
    Vogel* p_vogel = &papagei;
    Papagei* p_papagei = &papagei;
    p_tier->laufe();     // Ausgabe:
    p_vogel->laufe();    // Ausgabe:
    p_papagei->laufe();  // Ausgabe:
    p_tier->fliege();    // Ausgabe:
    vogel.fliege();      // Ausgabe:
    p_vogel->fliege();   // Ausgabe:
    p_papagei->fliege(); // Ausgabe:
    p_tier->friss();     // Ausgabe:
    p_vogel->friss();    // Ausgabe:
    p_papagei->friss();  // Ausgabe:
    return 0;
}
```

Aufgabe 7: Exceptions (8 min)

Ergänzen sie das Programm sinnvoll an den gekennzeichneten Stellen (//hier)

```
#include <iostream>
#include <string>
#include <typeinfo>
#include <queue>
using namespace std;

class Tier {
public:
    virtual ~Tier();
};

class Haustier : public Tier {};
class Hauskatze : public Haustier {};
class WildesTier : public Tier {};
class Tiger : public WildesTier {};
class Axolotl : public Tier {};

// Eine Haustierpraxis laesst die Tiere in einem Wartezimmer
// in einer Warteschlange (queue) warten.
class Haustierpraxis {
    queue<Tier> wartezimmer;
public:
    // Ein Tier betritt das Wartezimmer
    void wartezimmerBetreten(Tier& t) {
        // Wenn das Tier ein Haustier ist, dann wird es
        // in die Warteschlange mit aufgenommen.
        if (dynamic_cast<Haustier*>(&t))
        {
            wartezimmer.push(t);
        } else {
            // Falls das Tier ein Tiger ist, wirf einen Tiger als Ausnahme
            // HIER
```

// Fortsetzung auf Seite 14

```
// Fortsetzug von Seite 13
            // Andernfalls wirf den String "Unbekanntes Tier"
            // HIER




        }
    }
};

// Liefert einen Zeiger auf das gewaehlte Tier zurueck
Tier* gewaehltesTier(char c) {
    switch (c) {
    case '1' : return new Hauskatze();
    case '2' : return new Tiger();
    case '3' : return new Axolotl();
    default : return new Tier();
    }
}

int main() {
    char c;
    cout << "Welches Tier betritt das Wartezimmer?" << endl;
    cout << "1 = Hauskatze" << endl;
    cout << "2 = Tiger" << endl;
    cout << "3 = Axolotl" << endl;
    cin >> c;
    Tier * tptr = gewaehltesTier(c);
    Haustierpraxis praxis;
    // Ausnahmeueberpruefung hier beginnen
    // HIER


        praxis.wartezimmerBetreten(*tptr);


    // WildesTier (auf)fangen und gefangenes Tier weiterwerfen
    // HIER









// Fortsetzung auf Seite 15
```

```
// Fortsetzung von Seite 14
    // alle Strings auffangen und ausgeben
    // HIER




    // alle uebrigen Ausnahmen auffangen und einen Text ausgeben
    // HIER




    return 0;
}
```