Informations générales

Examen de module	Examen écrit	Date : 25.03.2021
Nom du module : **Programmation orientée objet**	Numéro de module :	40050200
Durée de l'examen : **60 min + 15 min**	Examinateur : Welp	

Nom : _____ Prénom_____

Numéro de matricule : _____

Évaluation

Tâche	1	2	3	4	5					Total
Points à atteindre	11	12	11	9	14					57
Points obtenus										
Signature de l'examinateur				Points obtenus en					Note	

Tâche 1: Du C au C++

1. Il s'agit de créer une fonction `polynôme` qui calcule des polynômes linéaires ou quadratiques :
   - Polynôme linéaire : $a1*x + a0$
   - Polynôme quadratique : $a2*x2 + a1*x + a0$

   Les paramètres de la fonction doivent être la `valeur x`, les coefficients $a1$ et $a0$ et, en option, le nombre de points.
   $a2$ sont transmis.

   Résolvez ce problème en
   1. `polynôme` surchargé
   2. définir `le polynôme` comme une fonction avec des paramètres par défaut.

   Donnez les définitions des **fonctions** pour les deux solutions. (5 points)

2. Qu'est-ce que le fragment de programme suivant affiche à l'écran ? (6 points)

```
short n[2] = {100,100} ;
short &r = n[0] ;
short *p = &r
; r = r*2 ;
p++ ;
*p = n[0] + n[1] ;.
```

   Après avoir exécuté ces instructions, à quoi ressemble l'image de la mémoire pour les programmes 32 bits ? Indiquez dans l'illustration suivante quels emplacements de mémoire sont occupés par quelles variables et quel est le contenu des emplacements de mémoire.

Nom	Contenu	Adresse
		...
		2000
		2002
		2004
		2006
		2008
		2010
		2012
		2014
		2016
		2018
		2020
		2022
		...

Tâche 2: Classes et objets

Définissez une classe `Fracture` pour la représentation des fractions. La classe doit avoir les propriétés suivantes *(NB : laissez suffisamment de place dans la classe pour pouvoir faire des ajouts au fur et à mesure)* :

- Le numérateur et le dénominateur doivent être stockés dans la classe en tant qu'attributs entiers. Il ne doit pas être possible d'accéder directement au numérateur et au dénominateur d'une fraction. L'accès doit se faire via les méthodes setter et getter correspondantes. **Définissez (implémentez)** les méthodes comme des fonctions en ligne.
- La classe doit disposer de constricteurs appropriés pour l'initialisation d'`objets fractionnaires`. Un `objet fractionnaire` doit pouvoir être initialisé par l'indication d'un numérateur et d'un dénominateur ainsi que par un nombre entier. L'instanciation d'un objet sans paramètre doit également être possible. Dans ce cas, l'objet doit être initialisé avec la valeur 0 (*voir remarque*). **Définissez** les méthodes comme des fonctions en ligne.
- La classe doit mettre à disposition une méthode `getDecimalzahl()` qui renvoie la fraction sous forme de nombre décimal. **Définissez** la fonction en dehors de la classe.
- La classe doit disposer d'un attribut de classe dans lequel le nombre d'objets `de rupture` est enregistré (compteur d'objets). Le nombre d'`objets de rupture` doit pouvoir être interrogé via une méthode de classe. **Définissez** la méthode.
- De plus, l'opérateur **~** à un chiffre pour la classe `fraction` doit être surchargé, de sorte que la fraction puisse être réduite par le plus grand diviseur commun du numérateur et du dénominateur. **Déclarez** l'opérateur. *(voir note)* (12 points)

Remarque

La classe `Fracture` doit par exemple pouvoir être utilisée de la manière suivante

```
int main()
{
        Fraction
        b1(12,9) ;
        Fraction b2(5)
        ;
        Fracture b3 ;


        cout << "b1 = " << b1.getZaerhler() << "/" << b1.getNenner()
             << " = " << b1.getDecimal() << endl ;
        cout << "b2 = " << b2.getZaehler() << "/" << b2.getNenner()
             << " = " << b2. getDecimal () << endl ;
        cout << "b3 = " << b3.getZaerhler() << "/" << b3.getNenner()
             << " = " << b3. getDecimal () << endl ;

        b3 = ~b1 ;
        cout << "b1 = " << b1.getZaerhler() << "/" << b1.getNenner()
             << " = " << b1. getDecimal () << endl ;
        cout << "b3 = " << b3.getZaehler() << "/" << b3.getNenner() << " = "
             << b3. getDecimal () << endl ;
        cout << "Il y a un total de " << Rupture::getAnzObjets() <<" ruptures" << endl ;
}
```

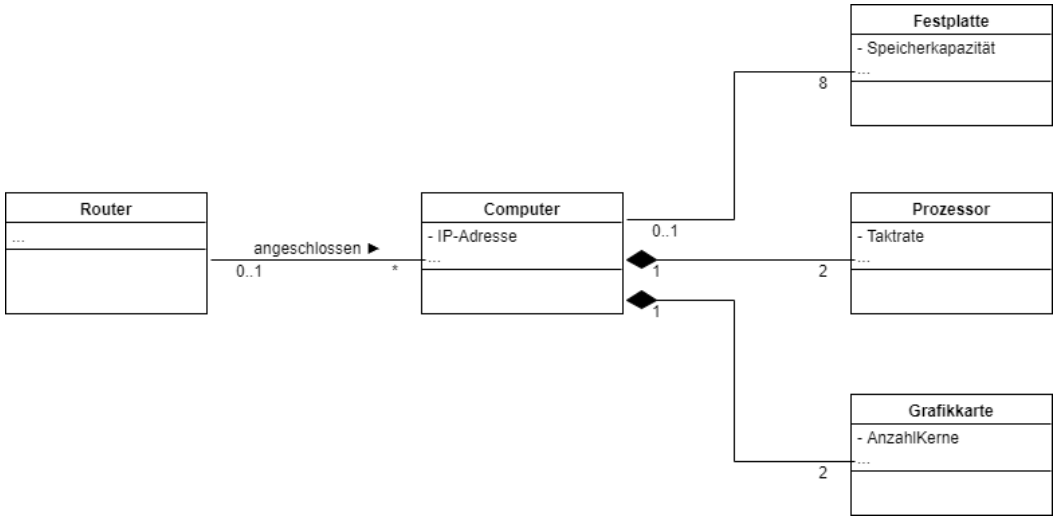
édition :

```
b1 = 12/9 = 1.33333
b2 = 5/1 = 5
b3 = 0/1 = 0
```

```
b1 = 4/3 = 1.33333
b3 = 4/3 = 1.33333
Il y a en tout 3 ruptures
```

Tâche 3 : Relations entre objets et modèles

Dans un logiciel de gestion de réseau, il existe les classes `routeur`, `ordinateur`, `disque dur`, `processeur` et `carte graphique`, qui sont en relation les unes avec les autres selon le diagramme de classes suivant.

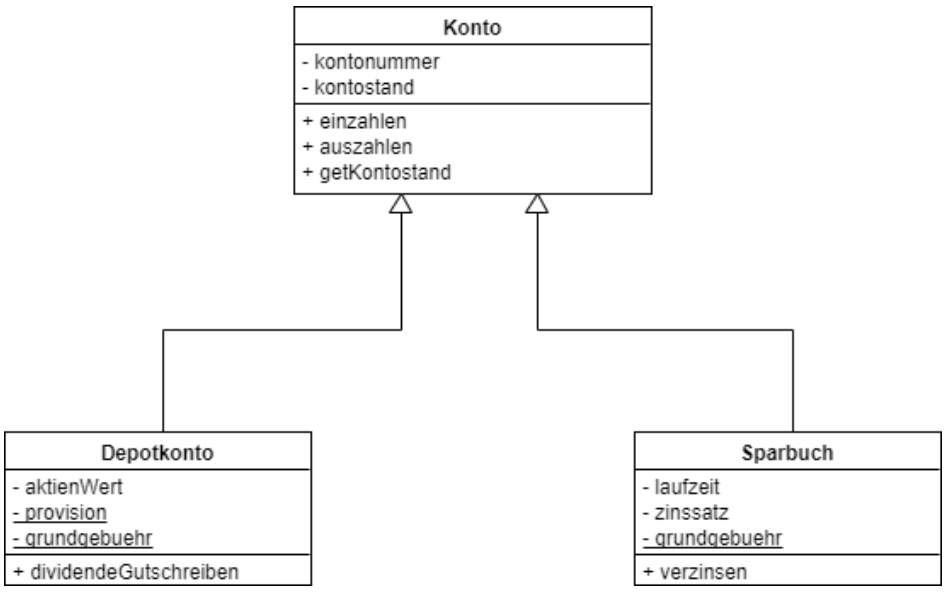


1. Quels attributs sont nécessaires dans la classe `Ordinateur` pour implémenter les relations avec les classes `Routeur`, `Disque dur`, `Processeur` et `Carte graphique` ? Répondez à la question en donnant la définition de classe pour la classe Ordinateur (attributs uniquement, les méthodes ne sont pas nécessaires). (4 points)

2. Donnez la définition de l'attribut de la classe `Router` qui permet de gérer la relation avec les objets informatiques (*connectés*). **Utilisez pour cela une classe conteneur de la Standard Template Library (STL).** (2 points)
   Définissez pour la classe `Router`, en utilisant l'attribut défini précédemment, une méthode qui permet de définir que le routeur est connecté à un ordinateur et une méthode qui renvoie tous les ordinateurs connectés au routeur. (5 points)

Tâche 4 : Héritage

Soit le diagramme de classes suivant :



La classe `Compte` est définie comme suit :

```
classe Compte
{
    unsigned int numéro de
    compte ; float solde de
    compte ;
public :
    Compte(unsigned int knr, float ks=0.0.) ;
    float getSoldeCompte() ;
    void déposer(float b) ;
    void payer(float b) ;
} ;
```

1. Donnez une définition de classe pour la classe `Compte de dépôt`. (3 points)
2. De quels attributs disposent **les objets** de la classe `Compte de dépôt` ? (3 points)
3. De quelles méthodes disposent **les objets** de la classe `Compte de dépôt` ? (2 points)

Tâche 5 : Modèles de conception

Soit une classe `Villes`, qui fournit de nombreuses informations sur les villes, et une classe `Carte`, qui peut représenter des pays, des villes, des fleuves, etc. sous forme de carte.

1. Dans un programme, vous voulez relier les deux classes `villes` et `carte` à l'aide du pattern Observer. Quelle classe joue le rôle d'observateur et quelle classe celui de sujet ? (2 points)
2. Pour implémenter le pattern Observer, vous disposez des deux classes générales `Sujet` et `Observateur`. Comment associez-vous ces deux classes aux classes `villes` et `carte` ? (2 points)
3. La classe `Beobachter` contient une méthode purement virtuelle `void actualiser()`. Comment cette méthode doit-elle être déclaréedans `Beobachter`? (2points)
4. La classe `Observateur` devient une classe abstraite grâce à la méthode purement virtuelle `Actualiser`. Quelles en sont les conséquences ? (3 points)
5. Quelle affirmation est correcte ? (1 point)
    a. La classe `des villes` dépend de la classe de `la carte`.
    b. La classe `carte` dépend de la classe `villes`.
6. Par quel mécanisme Qt le modèle Observer peut-il être facilement mis en œuvre, sans `classes` explicites de `sujet` et `d'observateur` ? Qu'est-ce qui doit être présent (c'est-à-dire déclaré) dans les classes `Staedte` et `Karte` ? Comment s'appelle la méthode qui permet de relier les objets de la classe `Staedte` aux objets de la classe `Card` ? (4 points)