Programmation Multi Paradigm en C++

PMP 2020-2021

Prénom : Ryana

Nom : KARAKi

Les questions peuvent avoir entre 0 et N réponses correctes où N est le nombre de réponses proposées. Les réponses correctes rapportent des points, les réponses incorrectes retranchent des points. Une question non répondue doit être barrée. Une ligne non cochée mais dont la réponse était correcte dans une question répondue retranche des points. Si une justification est demandée, celle ci rapporte des points si elle est correcte, et elle en retranche si elle est incorrecte ou non renseignée.

Vous répondrez sur la feuille directement

Q1. Quelles sont les caractéristiques d'une classe abstraite : 2+

- ☐ elle n'a pas de constructeur
- ☒ elle ne peut pas être instanciée
- ☒ au moins une de ses méthodes n'est pas définie (virtuelle pure)
- ☐ elle n'a pas de destructeur
- ☐ elle ne peut pas hériter d'une autre classe

soit le code suivant,

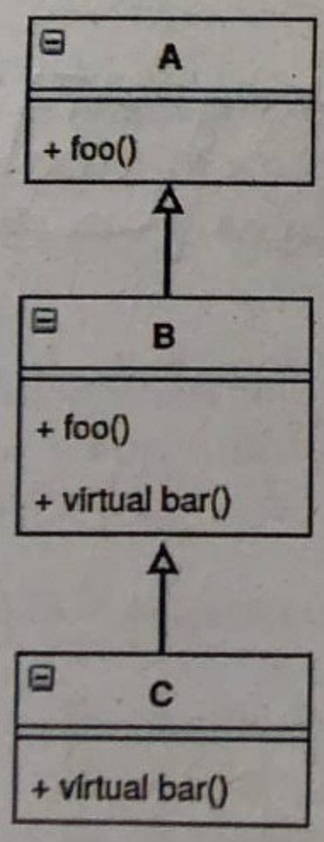
```
class A {
 public: A() { ... }
};
class B {
 public: B() { ... }
};
class C: public A, public B {
  public: C() { ... }
};
```

```
int main()
{
  C c;
  return 0;
}
```

***Q2**. Quel constructeur de classe est appelé ?* 1+ 2-

- ☐ A::A()
- ☐ B::B()
- ☒ C::C()
- ☐ on ne peut pas savoir

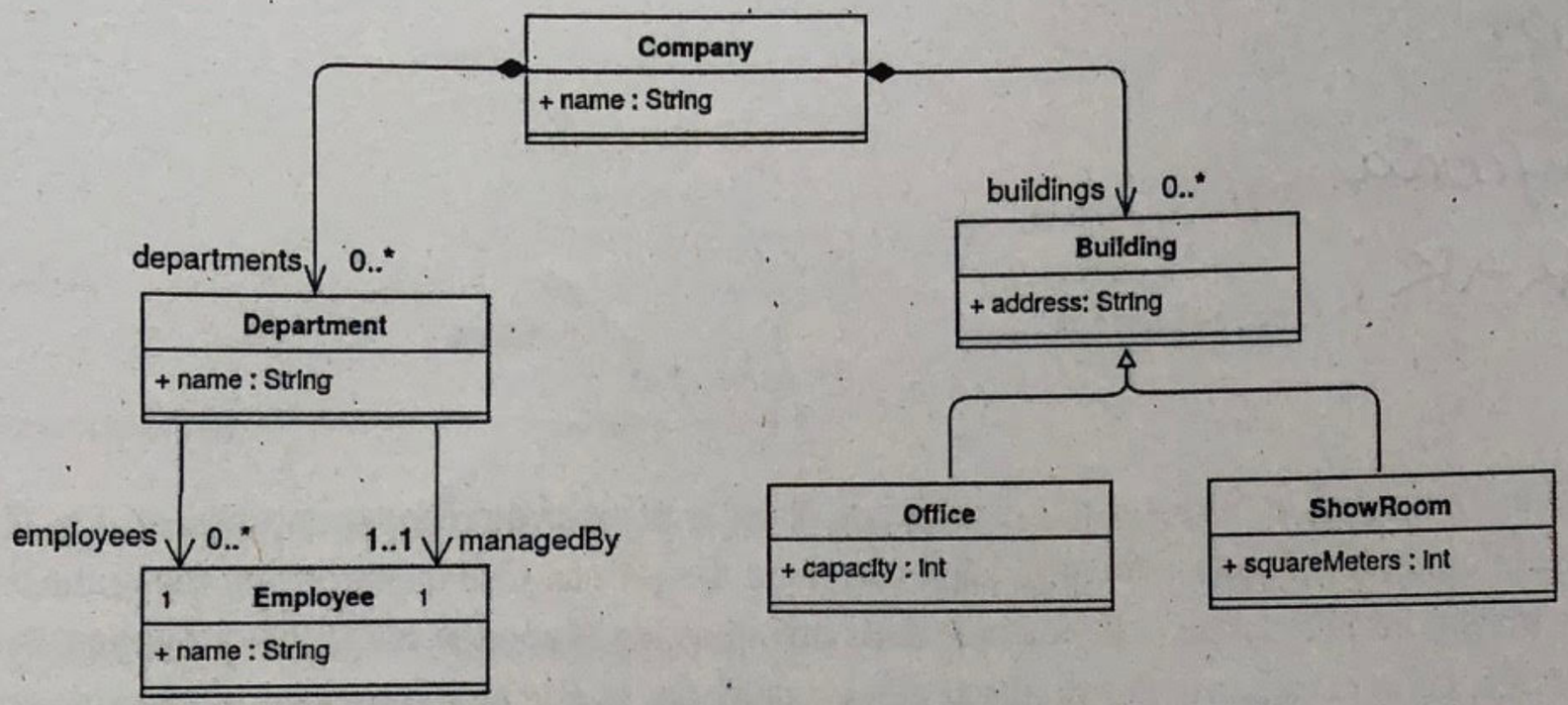
Soit le diagramme de classe suivant (les constructeurs ne sont pas représentés) :



Q3. Cocher la ligne si le commentaire du code est correct 6+3-

- ☒ A a1 ; //appel au constructeur de A
- ☒ B b1 ; //appel au constructeur de B
- ☒ C c1 = {18} ; //appel au constructeur de C
- ☒ B& b2 = c1 ; //appel au constructeur de B
- ☐ A* a2 = b2 ; //appel au constructeur de A
- ☒ A& a3 = b1 ; //appel au constructeur de A
- ☒ a1.foo() ; //appel à A::foo()
- ☐ a2.foo() ; //appel à A::foo() a2->foo()
- ☒ a3.bar() ; //appel à B::bar()
- ☒ b2.bar() ; //appel à C::bar()
- ☐ c1.foo() ; //erreur
- ☒ a2.bar() ; //erreur
- ☐ a2.bar() ; //appel à B::bar()
- ☐ a3.foo() ; //appel à B::foo()

Soit le diagramme de classe suivant:



Q4. Quel type choisiriez vous pour l'attribut *departments* dans la classe *Company*

- ☐ collection d'objets
- ☒ collection de pointeurs
- ☐ collection de références

KO

→ justification et implication : ..... La multiplicité entre Company et Department indique qu'on ne peut pas avoir une collection de références. De plus, Company est responsable du cycle de vie de Departments.

Q5. Quel type choisiriez vous pour l'attribut *managedBy* dans la classe ~~Company~~ Department

- ☐ objet
- ☐ pointeur
- ☒ référence

O

→ justification et implication : managedBy ne peut pas être nul, ce qui supprime l'option du pointeur et nous fait pencher pour l'option de la référence, car une référence ne peut pas être nulle.

Q6. Quel type choisiriez vous pour l'attribut *buildings* dans la classe *Company*

- ☐ collection d'objets
- ☒ collection de pointeurs
- ☐ collection de références

KO

→ justification et implication : La relation entre Company et Building est la même qu'entre Company et Departments. On peut donc choisir un type similaire pour les raisons expliquées plus haut.

Q7. Quelles fonctions de classe s'appellent des fonctions inline ?

- ☐ les fonctions déclarées à l'intérieur d'une classe
- ☐ les fonctions définies en dehors de la classe
- ☒ les fonctions définies à l'intérieur d'une classe
- ☐ les fonctions définies avec le mot clé *inline*
- ☐ les fonctions accédant aux membres statiques de la classe
- ☐ les fonctions tenant sur une seule ligne
- ☐ les fonctions bien alignées

1 + 1 −

Soit le code suivant

```
int main()
{
  Point *p1, *p2;
  p1 = new Point(5, 10);
  p2 = new Point(*p1);
  Point p3 = *p1;
  Point p4;
  p4 = p3;
  delete p1;
  return 0;
}
```

Q8. Combien de fois est appelé le constructeur de copie de la classe Point ?

.....

Q9. Combien de fois est appelé le constructeur d'initialisation de la classe Point ?

.....

Q10. y a t'il une fuite mémoire ?

- [x] oui
- [ ] non

OK

justification : Le pointeur p2 est alloué dynamiquement, donc il faut delete p2;

Q11. Qu'est-ce qu'une fonction virtuelle ?

- [ ] Une fonction que l'on n'est pas obligée d'implémenter
- [x] Une fonction qui rend une classe abstraite
- [ ] Une fonction qui manque de concret
- [ ] Une fonction qui est susceptible d'exister mais qui reste sans effet dans le présent

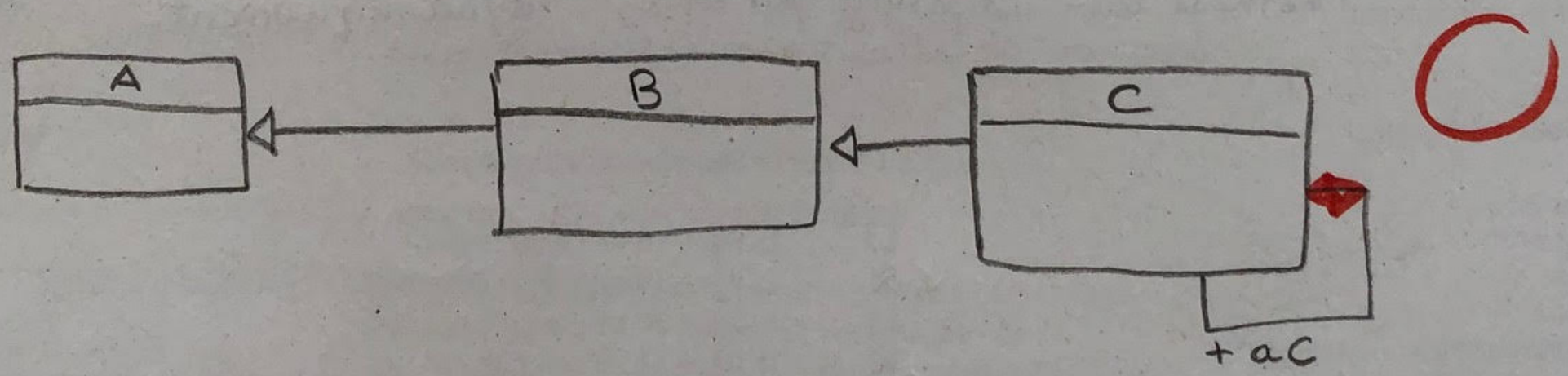
OK

Q12. Soit trois classe A, B, C. C hérite de B, B hérite de A. Quel est/sont la/les déclaration(s) possible(s) de la fonction *clone* dans la classe C ?

- [ ] `virtual void C::clone(C*);`
- [ ] `A* C::clone(C*);`
- [ ] `virtual A* C::clone();`
- [ ] `A* C::clone();`
- [ ] `virtual A* C::clone();`
- [x] `C* C::clone();`
- [x] `virtual C* C::clone();`

2 + 1 -

Q13. Dessiner un diagramme de classe dans lequel interviennent entre autres les classes A,B et C de la question précédente et où la fonction *clone* serait obligatoirement utilisée. Expliquer très succinctement.



O

Explications : Si la classe C a un attribut pointeur sur un autre objet de classe C, un setter de ce pointeur nécessite un clone. Un constructeur par copie de C a également besoin d'un clone.

Q14. À quoi sert une fonction clone ?

- [ ] À allouer de la mémoire dynamiquement (sur le tas)
- [x] À faire une copie polymorphique
- [x] À copier un objet

1 + 1 -

Soit le code suivant:

```
class Horaire {
 private:
  string jour;
  double heure;
 public:
  Horaire(string j, double h)
  : jour(j), heure(h){}

  void setHeure(double uneHeure){
   heure = uneHeure;
  }
};
```

```
class Examen {
  private:
    string nom;
    Horaire horaire;
    Examen (string n, Horaire h)
      : nom(n), horaire(h){}
  public:
    Horaire getHoraire(){
        return horaire;
    }
};
```

```
int main(){
  Examen cpp{"PMP"};
  cpp.setHoraire("Lundi", 09.00);
  Horaire hor = cpp.getHoraire().setHeure(23);
  sendToStudent(cpp);
}
```

Q15. À quelle heure est planifié l'examen envoyé aux étudiants ?

- ☒ À 09:00
- ☐ À 23:00
- ☐ On ne peut pas savoir.

Justifications :....

Soit le code suivant,

```
class A{
 //something
};
class B: public A{
public:
 //something
 A& doANew();
 void destroy(A a);
};
```

```
A& B::doANew(){
 return *(new A());
}

void B::destroy(A a){
 delete &a;
}
```

```
int main(){
 B* b;
 A& anA = b->create();
 A a2 = anA;
 b->destroy(anA);
return 0;
}
```

doANew();

Q16. Y a t'il une fuite mémoire

- ☒ oui
- ☐ non
- ☐ On ne peut pas savoir.

Justifications:

OK

La méthode doANew() alloue de la mémoire dynamiquement - La méthode destroy n'est pas un destructeur : c'est une méthode avec un passage par copie, donc on ne détruit pas réellement anA.

Soit le code suivant:

```
1   class A {
2     public: int id = 0;
3   };
4   class B {
5     int x = 1;
6     A a;
7   public:
8     B(string s) {//something}
9     operator A() { return a; }
10    operator int() { return x; }
11  };
12
13  void f1 ( int x ) { cout << x << endl; }
14  void f2 ( A a ) { cout << a.id << endl; }
15  void f3 (B b) {cout << "this is the end" << endl;}
16
17  int main() {
18    B b1;
19    B b2("toto");
20    f1(b2);
21    f2(b2);
22    string s("zaza");
23    f3(s);
24    return 0;
25  }
```

Q17. Cocher la ligne lorsqu'elle est vraie

- ☐ la ligne 18 ne compile pas
- ☒ la ligne 18 appelle le constructeur synthétisé par défaut
- ☐ la ligne 20 ne compile pas
- ☐ la ligne 21 ne compile pas
- ☒ la ligne 20 affiche la valeur de *x* de *b2*
- ☒ la ligne 21 affiche la valeur de *a.id* de *b2*
- ☐ la ligne 20 affiche un nombre quelconque (non connu)
- ☐ la ligne 21 affiche un nombre quelconque (non connu)
- ☒ la ligne 23 ne compile pas
- ☐ la ligne 23 fait un segmentation fault
- ☐ la ligne 23 affiche « *this is the end* »
- ☐ il y a une fuite mémoire

2+2