Programmation Multi Paradigmes en C++

PMP 2021-2022

Prénom :..Marcel.....

Nom :..Marceau--Locoste.....

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. Dans une question répondue, une ligne non cochée mais dont la réponse était correcte 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. Si le code ne compile pas, commentez les lignes incriminées et continuez en considérant qu'elles ont toujours été commentées. Si c'est demandé, elles seront alors considérées comme renvoyant faux.

Vous répondrez sur la feuille directement

Q1 : Quelles sont les caractéristiques d'une classe abstraite :

- [x] elle n'a pas de constructeur
- [ ] elle ne peut pas hériter d'une autre classe
- [x] elle ne peut pas être instanciée
- [x] au moins une de ses méthodes n'est pas définie
- [x] elle n'a pas de destructeur

2+ 2-

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 : Cochez la ligne si constructeur de classe est effectivement appelé ?

- [x] A::A()
- [x] B::B()
- [x] C::C()
- [ ] on ne peut pas savoir

3+ 0-

Soit le code suivant :

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

Q3 : Combien de fois est appelé l'opérateur d'affectation de la classe Point ? ..1 fois

Q4 : Combien de fois est appelé le constructeur de copie de la classe Point ? ...1 fois

Q5 : Combien de fois est appelé le constructeur d'initialisation de la classe Point ?..1 fois

Q6 : y a t'il une fuite mémoire ?

- [x] oui
- [ ] non

4+ 0-

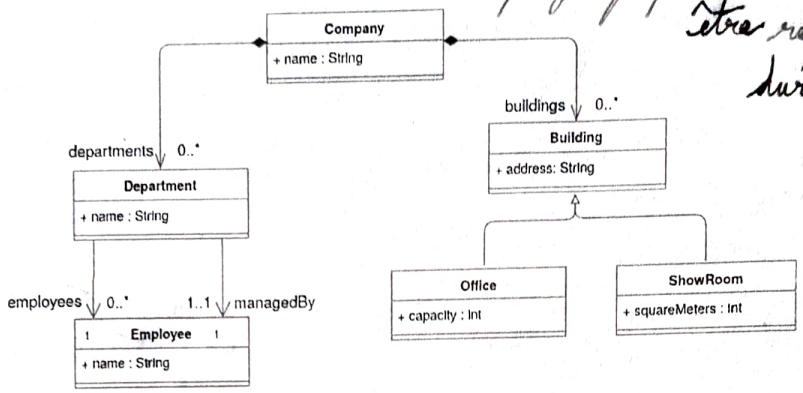
1+0−

Q7 : À quoi sert une fonction clone ?

- ☐ À allouer de la mémoire dynamiquement (sur le tas)
- ■ À faire une copie polymorphique
- ☐ À copier un objet

Alloue de la mémoire dynamiquement (sur le tas), mais ne sert pas à ça directement (se fait de cette manière afin d'avoir une copie polymorphique et de pouvoir ainsi être responsable de la durée de vie de l'objet copié)

Soit le diagramme de classe suivant:



Q8 : Quel type choisiriez vous pour l'attribut *departments* dans la classe *Company*

1+0−

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

→ justification et implication : On peut difficilement faire une collection de références c'est ce qu'il est conseillé.

Company doit être responsable du cycle de vie des départements avec la composition. C'est ainsi simple avec une collection d'objets.

Q9 : Quel type choisiriez vous pour l'attribut *managedBy* dans la classe *Department*

1+0−

- ☐ objet
- ■ pointeur
- ☐ référence

→ justification et implication : Un objet recquierait un employé et Department n'est pas responsable du cycle de vie de l'employé.

Une référence ne permettrait pas de changer l'attribut managed By (sauf si reconstruction de l'arborescence, ce qu'il ne faut pas faire)*

Q10 : Quel type choisiriez vous pour l'attribut *buildings* dans la classe *Company*

1+0

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

→ justification et implication : Collection de références trop compliquée voire impossible. Les pointeurs permettront de ne pas transtyper l'objet et donc de garder le type dynamique de celui-ci (car Building peut être Office ou ShowRoom).

Q11 : Pour quelle(s) classe(s) le constructeur de copie doit-il obligatoirement être redéfini ?

1+1−

- ☐ *Department*
- ■ *Company*
- ■ *Building*

* Il faudra bien sûr veiller à ce que le pointeur ne soit jamais null.

(fin de Q9) ✓

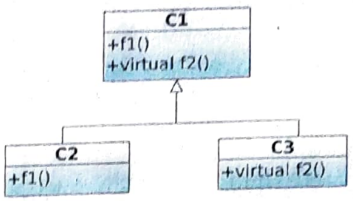
Q12 : Soit le diagramme de classe C++ de la figure ci dessous et le code suivant, donnez le nom qualifié des fonctions membres appelées (*e.g.*, C1::f1()).

```
C1 c1;
C2 c2;
C3 c3;
C1& a= c1;
C1* b=&c2;
C1  c= c3;
```

répondre ci dessous ↓

12+0-

```
c1.f1();    // C1::f1()
c2.f1();    // C2::f1()
c3.f1();    // C1::f1()
a.f1();     // C1::f1()
b->f1();    // C1::f1()
c.f1();     // C1::f1()
c1.f2();    // C1::f2()
c2.f2();    // C1::f2()
c3.f2();    // C3::f2()
a.f2();     // C1::f2()
b->f2();    // C1::f2()
c.f2();     // C1::f2()
```



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

```
int main(){
  Examen cpp{"PMP", {"Lundi", 23.00}};
  Horaire hor = cpp.getHoraire();
  hor.setHeure(14);
  sendToStudent(cpp);
}
```

Q13 : À quelle heure est planifié l'examen envoyé aux étudiants ?

- ☐ À 14:00
- ■ À 23:00
- ☐ On ne peut pas savoir.

1+0-

Soit le code suivant :

```
class A{
};

class B: public A{
public:
 A& doANewA();
 void destroy(A& a);
};
```

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

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

```
int main(){
 B b;
 A& anA = b.doANewA();
 A a2 = anA;
 b.destroy(anA);
return 0;
}
```

Q14 : Y a t'il une fuite mémoire

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

1+0-