

MOOC Intro POO C++

Corrigés semaine 6

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 20 : Animaux en peluche

Cet exercice correspond à l'exercice n°63 (pages 160 et 351) de l'ouvrage *C++ par la pratique* (3^e édition, PPUR).

La classe Animal ne présente aucune difficulté :

```
1 class Animal {
2 public:
3     void affiche() const;
4 protected:
5     string nom;
6     string continent;
7 };
8
9 void Animal::affiche() const {
10     cout << "Je suis un " << nom << " et je vis en " << continent << endl;
11 }
```

pas plus que les deux autres classes :

```
1 class EnDanger {
2 public:
3     void affiche() const;
4 protected:
5     unsigned int nombre;
6 };
7
8 void EnDanger::affiche() const {
9     cout << "Il ne reste que " << nombre
10         << " individus de mon espece sur Terre" << endl;
11 }
12
13 class Gadget {
14 public:
15     void affiche() const;
16     void affiche_prix() const;
17 protected:
18     string nom;
19     double prix;
20 };
21
22 void Gadget::affiche() const {
23     cout << "Mon nom est " << nom << endl;
24 }
25
```

```

26 void Gadget::affiche_prix() const {
27     cout << "Achetez-moi pour " << prix
28         << " francs et vous contribuerez a me sauver!" << endl;
29 }

```

L'ajout des constructeurs et destructeurs se fait aussi trivialement :

```

1  class Animal {
2  public:
3      Animal(string, string);
4      ~Animal();
5      void affiche() const;
6  protected:
7      string nom;
8      string continent;
9  };
10
11 void Animal::affiche() const {
12     cout << "Je suis un " << nom << " et je vis en " << continent << endl;
13 }
14
15 Animal::Animal(string nom, string continent)
16     : nom(nom), continent(continent)
17 {
18     cout << "Nouvel animal protege" << endl;
19 }
20
21 Animal::~~Animal() {
22     cout << "Je ne suis plus protege" << endl;
23 }
24
25 // *****
26 class EnDanger {
27 public:
28     void affiche() const;
29     EnDanger(unsigned int);
30     ~EnDanger();
31 protected:
32     unsigned int nombre;
33 };
34
35 EnDanger::EnDanger(unsigned int nombre)
36     : nombre(nombre)
37 {
38     cout << "Nouvel animal en danger" << endl;
39 }
40
41 EnDanger::~~EnDanger() {
42     cout << "ouf! je ne suis plus en danger" << endl;
43 }
44
45 void EnDanger::affiche() const {
46     cout << "Il ne reste que " << nombre
47         << " individus de mon espece sur Terre" << endl;
48 }
49
50 // *****
51 class Gadget {
52 public:
53     void affiche() const;
54     void affiche_prix() const;
55     Gadget(string, double);
56     ~Gadget();

```

```

57 protected:
58     string nom;
59     double prix;
60 };
61
62 Gadget::Gadget(string nom, double prix)
63     : nom(nom), prix(prix)
64 {
65     cout << "Nouveau gadget" << endl;
66 }
67
68 Gadget::~Gadget() {
69     cout << "Je ne suis plus un gadget" << endl;
70 }
71
72 void Gadget::affiche() const {
73     cout << "Mon nom est " << nom << endl;
74 }
75
76 void Gadget::affiche_prix() const {
77     cout << "Achetez-moi pour " << prix
78         << " francs et vous contribuerez a me sauver !"
79         << endl;
80 }

```

Définissez une classe Peluche héritant des classes Animal, EnDanger et Gadget :

```

1 class Peluche : public Animal, public EnDanger, public Gadget {
2 };

```

Dotez votre classe Peluche d'une méthode etiquette [...] codée au moyen des méthodes affiche et affiche_prix des super-classes :

```

1 class Peluche : public Animal, public EnDanger, public Gadget {
2 public:
3     void etiquette() const;
4 };
5
6 void Peluche::etiquette() const {
7     cout << "Hello," << endl;
8     Gadget::affiche();
9     Animal::affiche();
10    EnDanger::affiche();
11    affiche_prix();
12    cout << endl;
13 }

```

puis les constructeurs et destructeurs :

```

1 class Peluche : public Animal, public EnDanger, public Gadget {
2 public:
3     void etiquette() const;
4     Peluche(string, string, string, unsigned int, double);
5     ~Peluche();
6 };
7
8 Peluche::Peluche(string nom_animal, string nom_gadget,
9                 string continent, unsigned int nombre, double prix)
10    : Animal(nom_animal, continent), EnDanger(nombre),
11      Gadget(nom_gadget, prix)
12 { cout << "Nouvelle peluche" << endl; }
13
14 Peluche::~Peluche() {
15     cout << "Je ne suis plus une peluche" << endl;

```

16 }

Voici le code complet :

```

1 #include<iostream>
2 #include<string>
3 using namespace std;
4
5 // *****
6 class Animal {
7 public:
8     Animal(string, string);
9     ~Animal();
10    void affiche() const;
11 protected:
12    string nom;
13    string continent;
14 };
15
16 void Animal::affiche() const {
17     cout << "Je suis un " << nom << " et je vis en " << continent << endl;
18 }
19
20 Animal::Animal(string nom, string continent)
21 : nom(nom), continent(continent)
22 {
23     cout << "Nouvel animal protege" << endl;
24 }
25
26 Animal::~~Animal() {
27     cout << "Je ne suis plus protege" << endl;
28 }
29
30 // *****
31 class EnDanger {
32 public:
33     void affiche() const;
34     EnDanger(unsigned int);
35     ~EnDanger();
36 protected:
37     unsigned int nombre;
38 };
39
40 EnDanger::EnDanger(unsigned int nombre)
41 : nombre(nombre)
42 {
43     cout << "Nouvel animal en danger" << endl;
44 }
45
46 EnDanger::~~EnDanger() {
47     cout << "ouf! je ne suis plus en danger" << endl;
48 }
49
50 void EnDanger::affiche() const {
51     cout << "Il ne reste que " << nombre
52         << " individus de mon espece sur Terre" << endl;
53 }
54
55 // *****
56 class Gadget {
57 public:
58     void affiche() const;
59     void affiche_prix() const;

```

```

60     Gadget(string, double);
61     ~Gadget();
62 protected:
63     string nom;
64     double prix;
65 };
66
67 Gadget::Gadget(string nom, double prix)
68     : nom(nom), prix(prix)
69 {
70     cout << "Nouveau gadget" << endl;
71 }
72
73 Gadget::~Gadget() {
74     cout << "Je ne suis plus un gadget" << endl;
75 }
76
77 void Gadget::affiche() const {
78     cout << "Mon nom est " << nom << endl;
79 }
80
81 void Gadget::affiche_prix() const {
82     cout << "Achetez-moi pour " << prix
83         << " francs et vous contribuerez a me sauver !"
84         << endl;
85 }
86
87 // *****
88 class Peluche : public Animal, public EnDanger, public Gadget {
89 public:
90     void etiquette() const;
91     Peluche(string, string, string, unsigned int, double);
92     ~Peluche();
93 };
94
95 Peluche::Peluche(string nom_animal, string nom_gadget, string continent,
96     unsigned int nombre, double prix)
97     : Animal(nom_animal, continent), EnDanger(nombre), Gadget(nom_gadget, prix)
98 {
99     cout << "Nouvelle peluche" << endl;
100 }
101
102 Peluche::~Peluche() {
103     cout << "Je ne suis plus une peluche" << endl;
104 }
105
106 void Peluche::etiquette() const
107 {
108     cout << "Hello," << endl;
109     Gadget::affiche();
110     Animal::affiche();
111     EnDanger::affiche();
112     affiche_prix();
113     cout << endl;
114 }
115
116 // *****
117 int main()
118 {
119     Peluche panda ("Panda", "Ming", "Asie", 200, 20.0);
120     Peluche serpent("Cobra", "ssss", "Asie", 500, 10.0);
121     Peluche toucan ("Toucan", "Bello", "Amerique du Sud", 1000, 15.0);
122

```

```
123 panda.etiquette();
124 serpent.etiquette();
125 toucan.etiquette();
126
127 return 0;
128 }
```

Exercice 21 : Employés

Cet exercice correspond à l'exercice n°64 (pages 162 et 354) de l'ouvrage
C++ par la pratique (3^e édition, PPUR).

Codez une classe abstraite `Employe` [...] :

```

1 class Employe {
2 public:
3     virtual double calculer_salaire() const = 0;
4 protected:
5     string prenom;
6     string nom;
7     unsigned int age;
8     string date;
9 };

```

Dotez également votre classe d'un constructeur [...] et d'un destructeur virtuel vide.

```

1 class Employe {
2 public:
3     Employe(string prenom, string nom, unsigned int age, string date)
4         : nom(nom), prenom(prenom), age(age), date(date) {}
5     virtual ~Employe() {}
6     virtual double calculer_salaire() const = 0;
7 protected:
8     string prenom;
9     string nom;
10    unsigned int age;
11    string date;
12 };

```

Calcul du salaire

Codez une hiérarchie de classes pour les employés en respectant les conditions suivantes [...]

La première constatation que l'on peut faire est que les deux commerciaux (vendeur et représentant) ont une base commune de calcul. On peut créer une super-classe commune à ces deux classes.

Appelons-la par exemple `Commercial`. Elle reste une classe abstraite et il n'y a qu'à lui affecter l'attribut nécessaire (`chiffre_affaire`) et le constructeur qui va avec :

```

1 class Commercial: public Employe {
2 public:
3     Commercial(string prenom, string nom, unsigned int age, string date,
4                 double chiffre_affaire)
5         : Employe(prenom, nom, age, date), chiffre_affaire(chiffre_affaire)
6         {}
7     ~Commercial() {}
8 protected:
9     double chiffre_affaire;
10 };

```

On peut alors écrire les deux classes qui en héritent : `Vendeur` et `Representant` :

```

1 class Vendeur: public Commercial {
2 public:
3     Vendeur(string prenom, string nom, unsigned int age,
4             string date, double chiffre_affaire)
5         : Commercial(prenom, nom, age, date, chiffre_affaire)
6         {}
7     ~Vendeur() {}

```

```

8   double calculer_salaire() const;
9   string get_nom() const;
10  };
11
12  class Representant: public Commercial {
13  public:
14      Representant(string prenom, string nom, unsigned int age,
15                  string date, double chiffre_affaire)
16          : Commercial(prenom, nom, age, date, chiffre_affaire)
17      {}
18      ~Representant() {}
19      double calculer_salaire() const;
20      string get_nom() const;
21  };

```

Il n'y a plus qu'à spécifier leurs méthodes spécifiques :

```

1  double Vendeur::calculer_salaire() const {
2      return (0.2 * chiffre_affaire) + 400;
3  }
4
5  string Vendeur::get_nom() const {
6      return "Le vendeur " + prenom + ' ' + nom;
7  }
8
9  double Representant::calculer_salaire() const {
10     return (0.2 * chiffre_affaire) + 800;
11 }
12
13 string Representant::get_nom() const {
14     return "Le representant " + prenom + ' ' + nom;
15 }

```

Remarque. Pour les programmeurs plus avancés, il serait préférable de ne pas coder les constantes 400 et 800 «en dur», mais de définir des constantes statiques dans la classe. Pour un programme de petite taille cela ne change pas grand-chose, mais pour un projet d'envergure cela facilite la maintenance.

Les classes Technicien et Manutentionnaire se font de la même façon, sans difficulté :

```

1  class Technicien: public Employe {
2  public:
3      Technicien(string prenom, string nom, unsigned int age, string date,
4                  unsigned int unites)
5          : Employe(prenom, nom, age, date), unites(unites)
6      {}
7      ~Technicien() {}
8      double calculer_salaire() const;
9      string get_nom() const;
10  protected:
11      unsigned int unites;
12  };
13
14  double Technicien::calculer_salaire() const {
15      return 5.0 * unites;
16  }
17
18  string Technicien::get_nom() const {
19      return "Le technicien " + prenom + ' ' + nom;
20  }
21
22  class Manutentionnaire: public Employe {
23  public:
24      Manutentionnaire(string prenom, string nom, unsigned int age,

```

```

25         string date, unsigned int heures)
26     : Employe(prenom, nom, age, date), heures(heures)
27     {}
28     ~Manutentionnaire() {}
29     double calculer_salaire() const;
30     string get_nom() const;
31 protected:
32     unsigned int heures;
33 };
34
35 double Manutentionnaire::calculer_salaire() const {
36     return 65.0 * heures;
37 }
38
39 string Manutentionnaire::get_nom() const {
40     return "Le manut. " + prenom + ' ' + nom;
41 }

```

Employés à risques

Voici finalement l'héritage multiple. Mais avant il faut définir la super-classe d'employé à risque :

```

1 class ARisque {
2 public:
3     ARisque(double prime = 100) : prime(prime) {}
4     virtual ~ARisque() {}
5 protected:
6     double prime;
7 };

```

Concernant l'ordre d'héritage, il semble ici évident que ces employés sont avant tout des employés (avant d'être « à risque »). On a donc naturellement :

```

1 class TechnARisque: public Technicien, public ARisque {
2 };
3 class ManutARisque: public Manutentionnaire, public ARisque {
4 };

```

qu'il suffit ensuite de compléter des éléments habituels :

```

1 class TechnARisque: public Technicien, public ARisque {
2 public:
3     TechnARisque(string prenom, string nom, unsigned int age,
4                 string date, unsigned int unites, double prime)
5         : Technicien(prenom, nom, age, date, unites), ARisque(prime)
6     {}
7     double calculer_salaire() const;
8 };
9
10 double TechnARisque::calculer_salaire() const {
11     return Technicien::calculer_salaire() + prime;
12 }
13
14 class ManutARisque: public Manutentionnaire, public ARisque {
15 public:
16     ManutARisque(string prenom, string nom, unsigned int age,
17                 string date, unsigned int heures, double prime)
18         : Manutentionnaire(prenom, nom, age, date, heures), ARisque(prime)
19     {}
20     double calculer_salaire() const;
21 };
22

```

```

23 double ManutARisque::calculer_salaire() const {
24     return Manutentionnaire::calculer_salaire() + prime;
25 }

```

Collection d'employés

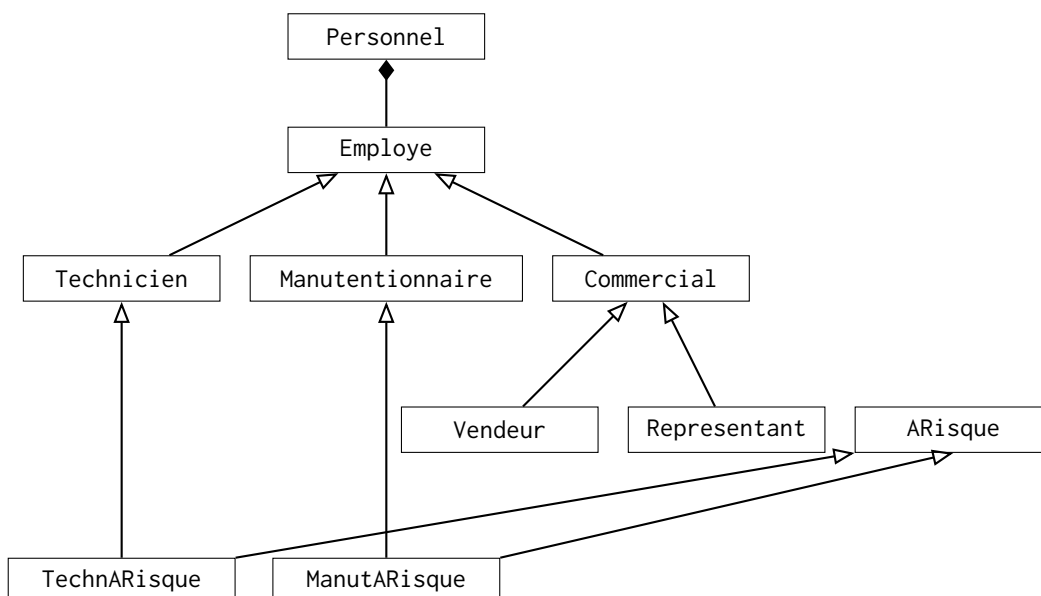
Il s'agit ici de quelque chose de très similaire à ce que nous avons fait la semaine dernière. Voici donc la solution :

```

1  class Personnel {
2  public:
3      void ajouter_employe(Employe* newbie) { staff.push_back(newbie); }
4      void licencie();
5      void afficher_salaires() const;
6      double salaire_moyen() const;
7  protected:
8      vector<Employe*> staff;
9  };
10
11 void Personnel::licencie() {
12     for (auto p : staff) delete p;
13     staff.clear();
14 }
15
16 double Personnel::salaire_moyen() const {
17     double somme(0.0);
18     for (auto p : staff) {
19         somme += p->calculer_salaire();
20     }
21     return somme / staff.size();
22 }
23
24 void Personnel::afficher_salaires() const {
25     for (auto p : staff) {
26         cout << p->get_nom() << " gagne "
27              << p->calculer_salaire() << " francs."
28              << endl;
29     }
30 }

```

Voici pour résumer, le diagramme d'héritage (et encapsulation) de cet exercice :



et le code source complet :

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4 using namespace std;
5
6 /* *****
7  * La classe Employe
8  */
9 class Employe {
10 public:
11     Employe(string prenom, string nom, unsigned int age, string date)
12         : nom(nom), prenom(prenom), age(age), date(date) {}
13     virtual ~Employe() {}
14     virtual double calculer_salaire() const = 0;
15     virtual string get_nom() const;
16 protected:
17     string nom;
18     string prenom;
19     unsigned int age;
20     string date;
21 };
22
23 string Employe::get_nom() const { return "L'employe " + prenom + ' ' + nom; }
24
25 /* *****
26  * La classe Commercial (factorise Vendeur et Representant)
27  */
28 class Commercial: public Employe {
29 public:
30     Commercial(string prenom, string nom, unsigned int age, string date,
31                 double chiffre_affaire)
32         : Employe(prenom, nom, age, date), chiffre_affaire(chiffre_affaire)
33     {}
34     ~Commercial() {}
35 protected:
36     double chiffre_affaire;
37 };
38
39 /* *****
40  * La classe Vendeur
41  */
42 class Vendeur: public Commercial {
43 public:
44     Vendeur(string prenom, string nom, unsigned int age, string date,
45             double chiffre_affaire)
46         : Commercial(prenom, nom, age, date, chiffre_affaire)
47     {}
48     ~Vendeur() {}
49     double calculer_salaire() const;
50     string get_nom() const;
51 };
52
53 double Vendeur::calculer_salaire() const {
54     return (0.2 * chiffre_affaire) + 400;
55 }
56
57 string Vendeur::get_nom() const { return "Le vendeur " + prenom + ' ' + nom; }
58
59 /* *****
60  * La classe Representant
61  */

```

```

62 class Representant: public Commercial {
63 public:
64     Representant(string prenom, string nom, unsigned int age, string date,
65                 double chiffre_affaire)
66         : Commercial(prenom, nom, age, date, chiffre_affaire)
67     {}
68     ~Representant() {}
69     double calculer_salaire() const;
70     string get_nom() const;
71 };
72
73 double Representant::calculer_salaire() const {
74     return (0.2 * chiffre_affaire) + 800;
75 }
76
77 string Representant::get_nom() const { return "Le representant " + prenom + ' ' + nom; }
78
79 /* *****
80 * La classe Technicien (Production)
81 */
82 class Technicien: public Employe {
83 public:
84     Technicien(string prenom, string nom, unsigned int age, string date,
85                unsigned int unites)
86         : Employe(prenom, nom, age, date), unites(unites)
87     {}
88     ~Technicien() {}
89     double calculer_salaire() const;
90     string get_nom() const;
91 protected:
92     unsigned int unites;
93 };
94
95 double Technicien::calculer_salaire() const {
96     return 5.0 * unites;
97 }
98
99 string Technicien::get_nom() const { return "Le technicien " + prenom + ' ' + nom; }
100
101 /* *****
102 * La classe Manutentionnaire
103 */
104 class Manutentionnaire: public Employe {
105 public:
106     Manutentionnaire(string prenom, string nom, unsigned int age, string date,
107                      unsigned int heures)
108         : Employe(prenom, nom, age, date), heures(heures)
109     {}
110     ~Manutentionnaire() {}
111     double calculer_salaire() const;
112     string get_nom() const;
113 protected:
114     unsigned int heures;
115 };
116
117 double Manutentionnaire::calculer_salaire() const {
118     return 65.0 * heures;
119 }
120
121 string Manutentionnaire::get_nom() const { return "Le manut. " + prenom + ' ' + nom; }
122
123 /* *****
124 * La classe d'employes a risque

```

```

125 */
126 class ARisque {
127 public:
128     ARisque(double prime = 100) : prime(prime) {}
129     virtual ~ARisque() {}
130 protected:
131     double prime;
132 };
133
134 /* *****
135 * Une premiere sous-classe d'employe a risque
136 */
137 class TechnARisque: public Technicien, public ARisque {
138 public:
139     TechnARisque(string prenom, string nom, unsigned int age, string date,
140                 unsigned int unites, double prime)
141         : Technicien(prenom, nom, age, date, unites), ARisque(prime)
142     {}
143     double calculer_salaire() const;
144 };
145
146 double TechnARisque::calculer_salaire() const {
147     return Technicien::calculer_salaire() + prime;
148 }
149
150 /* *****
151 * Une autre sous-classe d'employe a risque
152 */
153 class ManutARisque: public Manutentionnaire, public ARisque {
154 public:
155     ManutARisque(string prenom, string nom, unsigned int age, string date,
156                 unsigned int heures, double prime)
157         : Manutentionnaire(prenom, nom, age, date, heures), ARisque(prime)
158     {}
159     double calculer_salaire() const;
160 };
161
162 double ManutARisque::calculer_salaire() const {
163     return Manutentionnaire::calculer_salaire() + prime;
164 }
165
166 /* *****
167 * La classe Personnel
168 */
169 class Personnel {
170 public:
171     void ajouter_employe(Employe* newbie) { staff.push_back(newbie); }
172     void licencie();
173     void afficher_salaires() const;
174     double salaire_moyen() const;
175 protected:
176     vector<Employe*> staff;
177 };
178
179 void Personnel::licencie() {
180     for (unsigned int i(0); i < staff.size(); i++) {
181         delete staff[i];
182     }
183     staff.clear();
184 }
185
186 double Personnel::salaire_moyen() const {
187     double somme(0.0);

```

```
188     for (unsigned int i(0); i < staff.size(); i++) {
189         somme += staff[i]->calculer_salaire();
190     }
191     return somme / staff.size();
192 }
193
194 void Personnel::afficher_salaires() const {
195     for (unsigned int i(0); i < staff.size(); i++) {
196         cout << staff[i]->get_nom() << " gagne "
197             << staff[i]->calculer_salaire() << " francs."
198             << endl;
199     }
200 }
201
202 // =====
203 int main () {
204     Personnel p;
205     p.ajouter_employe(new Vendeur("Pierre", "Business", 45, "1995", 30000));
206     p.ajouter_employe(new Representant("Leon", "Vendtout", 25, "2001", 20000));
207     p.ajouter_employe(new Technicien("Yves", "Bosseur", 28, "1998", 1000));
208     p.ajouter_employe(new Manutentionnaire("Jeanne", "Stocketout", 32, "1998", 45));
209     p.ajouter_employe(new TechnARisque("Jean", "Flippe", 28, "2000", 1000, 200));
210     p.ajouter_employe(new ManutARisque("Al", "Abordage", 30, "2001", 45, 120));
211
212     p.afficher_salaires();
213     cout << "Le salaire moyen dans l'entreprise est de "
214         << p.salaire_moyen() << " francs." << endl;
215
216     // liberation memoire
217     p.licencie();
218 }
```

Exercice 22 : Jeu de cartes

Cet exercice correspond à l'exercice n°65 (pages 164 et 358) de l'ouvrage
C++ par la pratique (3^e édition, PPUR).

```

1 #include <iostream>
2 #include <vector>
3 #include <string>
4 using namespace std;
5
6 class Couleur {
7 public:
8     typedef enum { ROUGE, VERT, BLEU, BLANC, NOIR } Choix;
9     Couleur(Choix c) : valeur(c) {}
10    virtual ~Couleur() {}
11    Choix valeur;
12    void affiche(ostream&, bool feminin = false) const;
13 };
14
15 void Couleur::affiche(ostream& out, bool feminin) const {
16     switch (valeur) {
17     case ROUGE: out << "rouge"; break;
18     case VERT:
19         out << "vert";
20         if (feminin) out << 'e';
21         break;
22     case BLEU:
23         out << "bleu";
24         if (feminin) out << 'e';
25         break;
26     case BLANC:
27         out << "blanc";
28         if (feminin) out << "he";
29         break;
30     case NOIR:
31         out << "noir";
32         if (feminin) out << 'e';
33         break;
34     }
35 }
36
37 // -----
38 class Carte {
39 public:
40     Carte(unsigned int cost = 0) : cost(cost) {}
41     virtual ~Carte(){}
42     virtual void afficher(ostream& out) const {
43         out << "de cout " << cost; }
44
45 protected:
46     unsigned int cost;
47 };
48
49 ostream& operator<<(ostream& out, const Carte& c) {
50     c.afficher(out);
51     return out;
52 }
53
54 // -----
55 class Terrain : public virtual Carte {
56 public:

```

```

57 Terrain(Couleur c) : couleur(c) {
58     cout << "Un nouveau terrain." << endl;
59 }
60 virtual ~Terrain() {}
61 void afficher(ostream&) const;
62 protected:
63     Couleur couleur;
64 };
65
66 void Terrain::afficher(ostream& out) const {
67     out << "Un terrain ";
68     couleur.affiche(out);
69     out << "." << endl;
70 }
71
72 // -----
73 class Creature : public virtual Carte {
74 public:
75     Creature(unsigned int cost, string nom, unsigned int attaque,
76             unsigned int defense) :
77         Carte(cost), nom(nom), attaque(attaque), defense(defense) {
78         cout << "Une nouvelle creature." << endl;
79     }
80     virtual ~Creature() {}
81     void afficher(ostream&) const;
82 protected:
83     string nom;
84     unsigned int attaque;
85     unsigned int defense;
86 };
87
88 void Creature::afficher(ostream& out) const {
89     out << "Une creature " << nom << ' ' << attaque << "/"
90         << defense << ' ';
91     Carte::afficher(out);
92     out << endl;
93 }
94
95 // -----
96 class Sortilege : public virtual Carte {
97 public:
98     Sortilege(unsigned int cost, string nom, string desc) :
99         Carte(cost), nom(nom), description(desc) {
100         cout << "Un sortilege de plus." << endl;
101     }
102     virtual ~Sortilege() {}
103     void afficher(ostream&) const;
104 protected:
105     string nom;
106     string description;
107 };
108
109 void Sortilege::afficher(ostream& out) const {
110     out << "Un sortilege " << nom << ' ';
111     Carte::afficher(out);
112     out << endl;
113 }
114
115 // -----
116 class CreatureTerrain : public Creature, public Terrain {
117 public:
118     CreatureTerrain(unsigned int cost, string nom, unsigned int attaque,
119                    unsigned int defense, Couleur couleur)

```

```

120     : Carte(cost), Creature(cost, nom, attaque, defense),
121       Terrain(couleur)
122     {
123         cout << "Houla, une creature/terrain." << endl;
124     }
125     virtual ~CreatureTerrain() {}
126     void afficher(ostream& const);
127 };
128
129 void CreatureTerrain::afficher(ostream& out) const {
130     out << "Une creature/terrain ";
131     couleur.affiche(out, true);
132     out << ' ' << nom << ' ' << attaque << "/" << defense << ' ';
133     Carte::afficher(out);
134     out << endl;
135 }
136
137 // -----
138 class Jeu {
139 public:
140     Jeu(){ cout << "On change de main" << endl; }
141     virtual ~Jeu(){}
142     void jette();
143     void ajoute(Carte* carte) { contenu.push_back(carte); }
144 private:
145     vector<Carte*> contenu;
146     friend ostream& operator<<(ostream&, const Jeu&);
147 };
148
149 ostream& operator<<(ostream& out, const Jeu& j) {
150     for (auto carte : j.contenu)
151         out << " + " << *carte;
152     return out;
153 }
154
155 void Jeu::jette() {
156     cout << "Je jette ma main." << endl;
157     for (auto& carte : contenu) { delete carte; }
158     contenu.clear();
159 }
160
161 // -----
162 int main()
163 {
164     Jeu mamain;
165
166     mamain.ajoute(new Terrain(Couleur::BLEU));
167     mamain.ajoute(new Creature(6, "Golem", 4, 6));
168     mamain.ajoute(new Sortilege(1, "Croissance Gigantesque",
169         "La creature ciblee gagne +3/+3 jusqu'a la fin du tour"));
170     mamain.ajoute(new CreatureTerrain(2, "Ondine", 1, 1, Couleur::BLEU));
171
172     cout << "La, j'ai en stock :" << endl;
173     cout << mamain;
174
175     mamain.jette();
176
177     return 0;
178 }

```