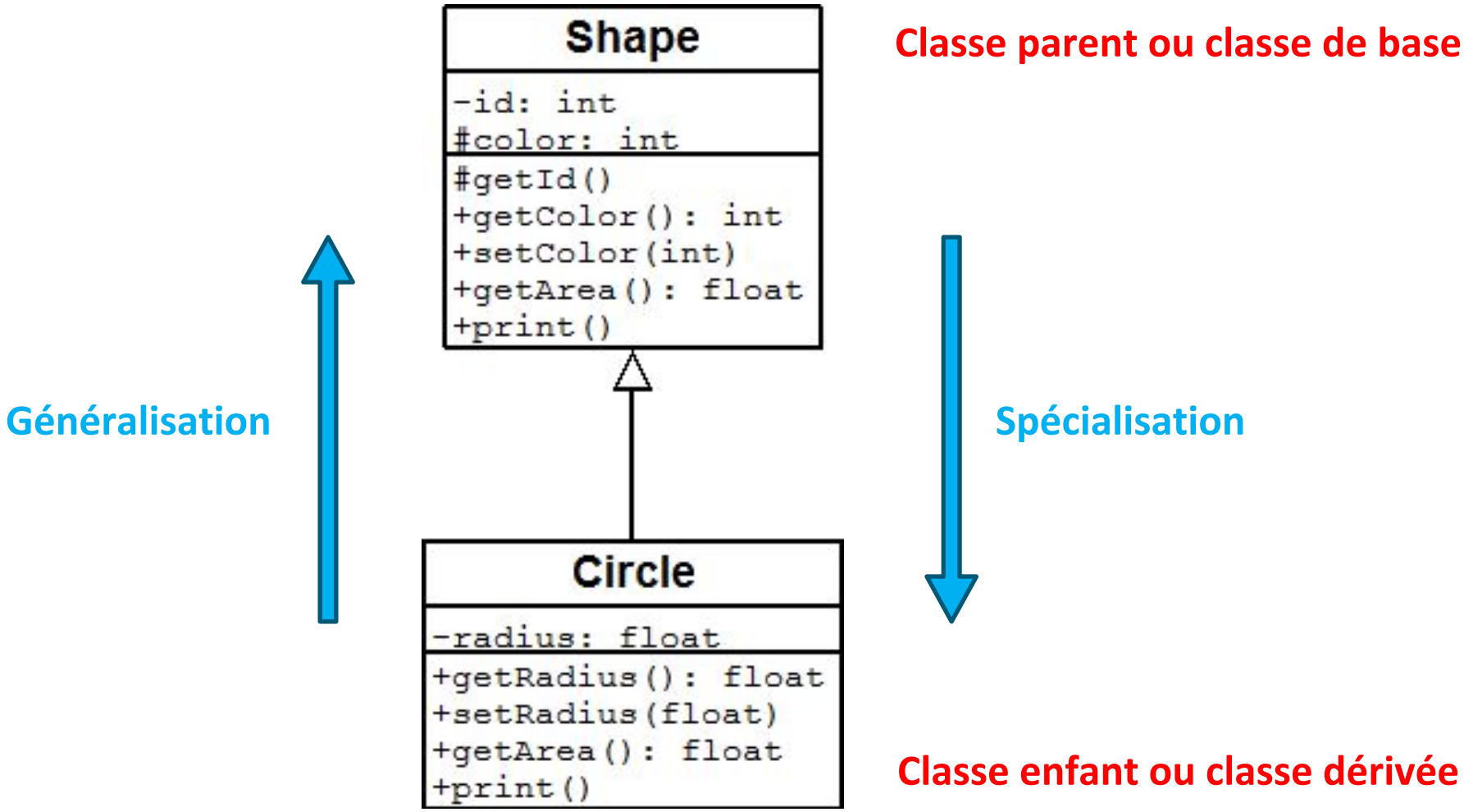
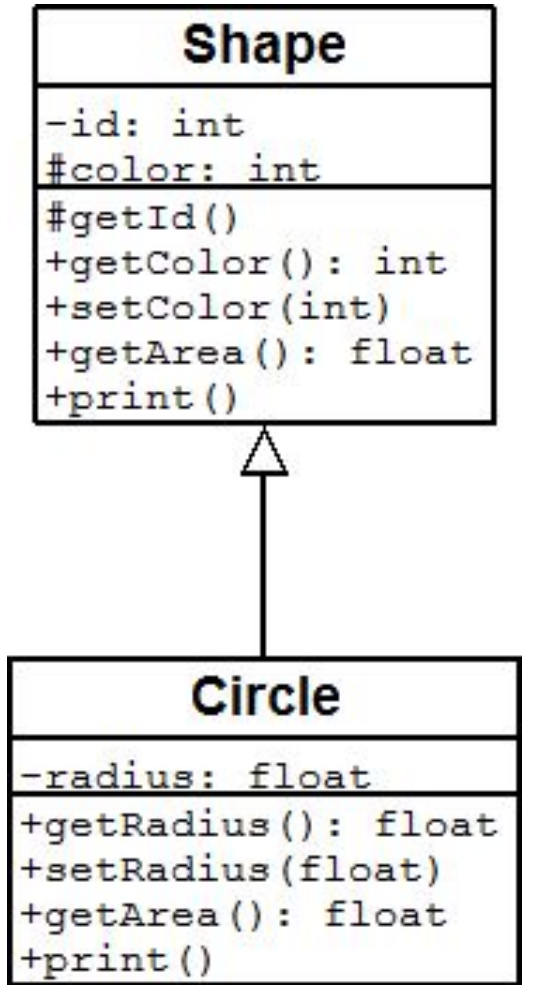
Principe de l'héritage



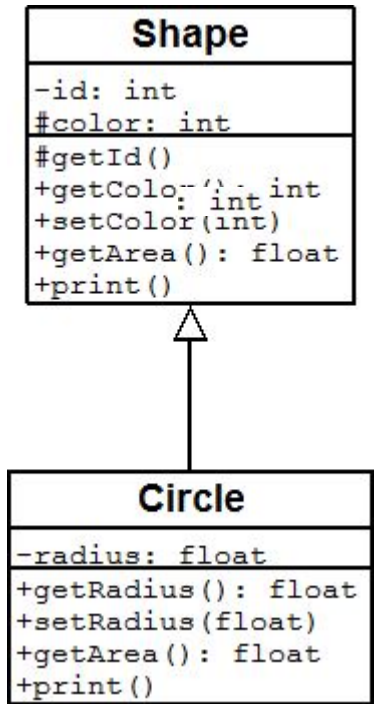
Principe de l'héritage

- La classe enfant :
  1. Hérite de **tous les membres** de la classe parent
  2. Accède seulement au membres **publiques** ou **protégés** du parent
  3. Peut **ajouter** de nouveaux membres
  4. Peut **réécrire** des méthodes de la classe parent

     (polymorphisme dynamique)



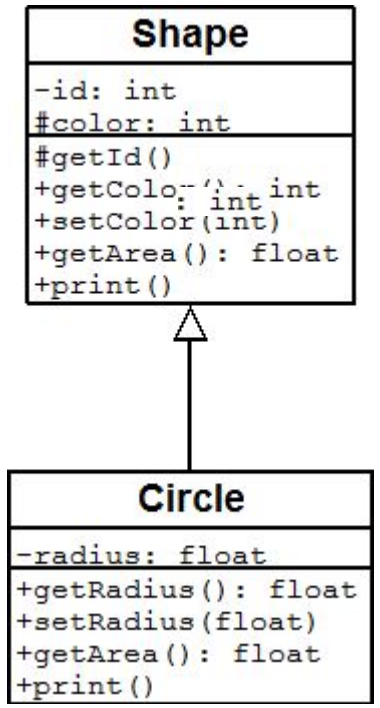
Syntaxe



Shape.h

```
class Shape
{
    private:
        int id;

    protected:
        int color;
        int getId();

    public:
        ... // constructors
        int getColor();
        void setColor();
        float getArea();
        void print();
};
```

Syntaxe



Circle.h

```
#include "Shape.h"

Class Circle : public Shape
{
    private:
        float radius;

    public:
        ... // constructors
        float getRadius();
        void
setRadius(float);
        float getArea();
        void print();
};
```

Héritage et accès aux membres

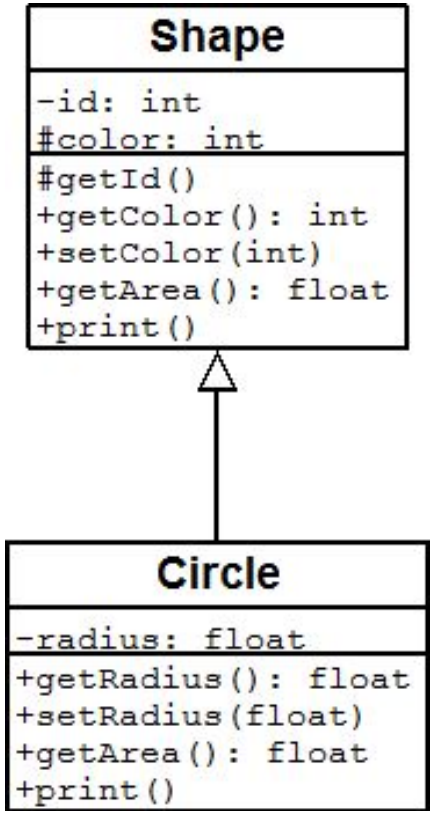
- La classe enfant hérite des membres de la classe parent
- Ce qui implique que la classe enfant connait les membres de la classe parent
- **Mais le contraire est faux**

Exemples

CircleTest.cpp

```
... // all required lines

int main()
{
    Shape s;

    s.setColor(1);

    return 0;
}
```

✔ Correct

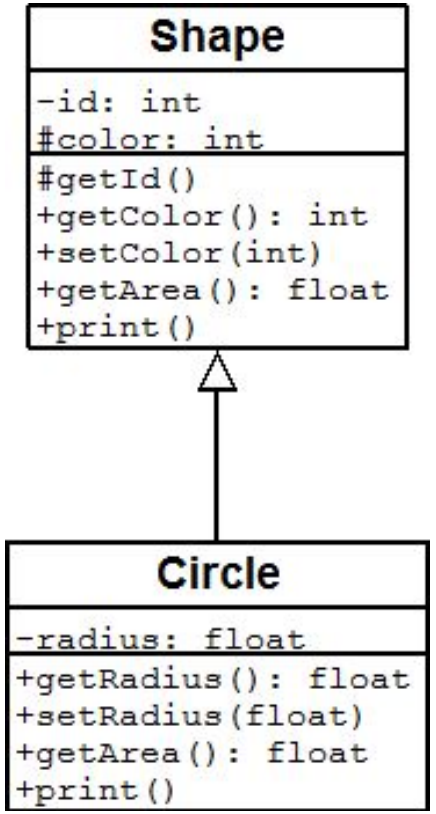


Exemples

CircleTest.cpp

```
... // all required lines

int main()
{
Circle c;

    c.setColor(1);

    return 0;
}
```

✔ Correct

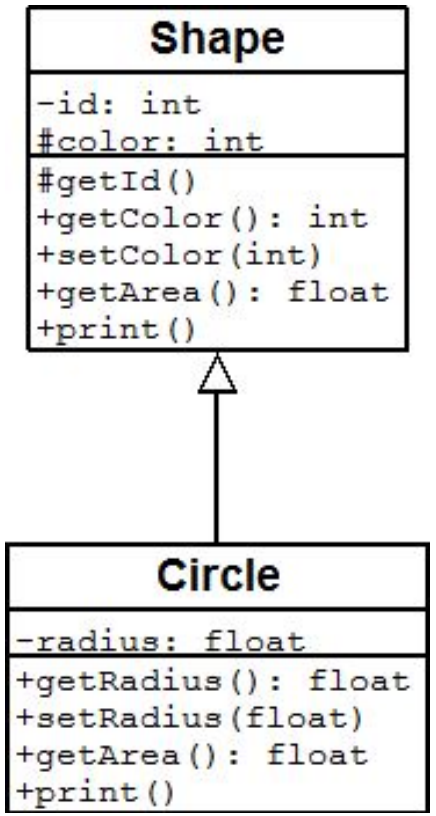


Exemples

CircleTest.cpp

```
... // all required lines

int main()
{
Circle c;

    c.setRadius(2.1);

    return 0;
}
```

✔ Correct

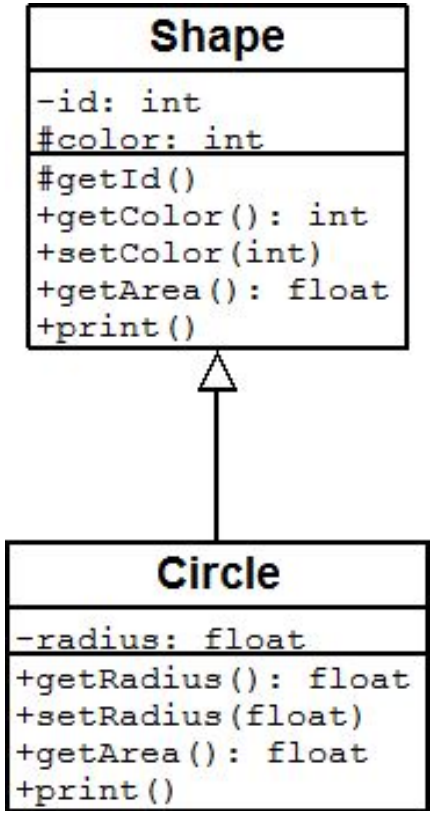


Exemples

CircleTest.cpp

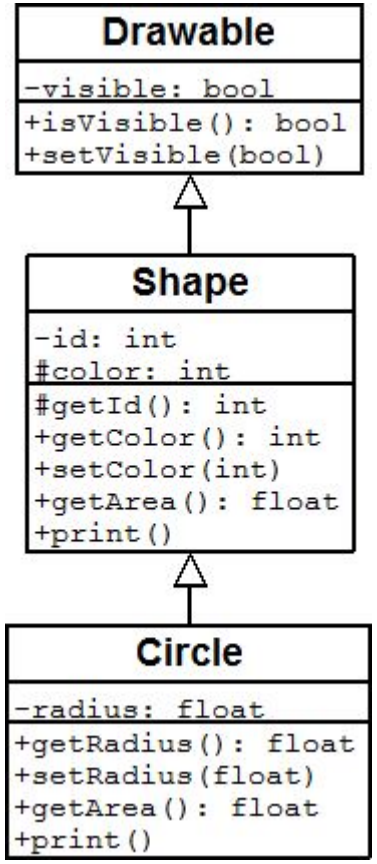
```
... // all required lines

int main()
{
Shape s;

s.setRadius(2.1);

    return 0;
}
```

✗ Incorrect



Transitivité de l'héritage

- L'héritage des membres est transitif:
  - Si Shape hérite de Drawable
  - Et Circle hérite de Shape
  - Alors Circle hérite de Drawable



- Une classe connait les membres de ses ancêtres, mais pas ceux de ses descendants.

- La recherche des membres se fait de l'enfant vers le parent, tant qu'il existe un parent.

Accès aux membres

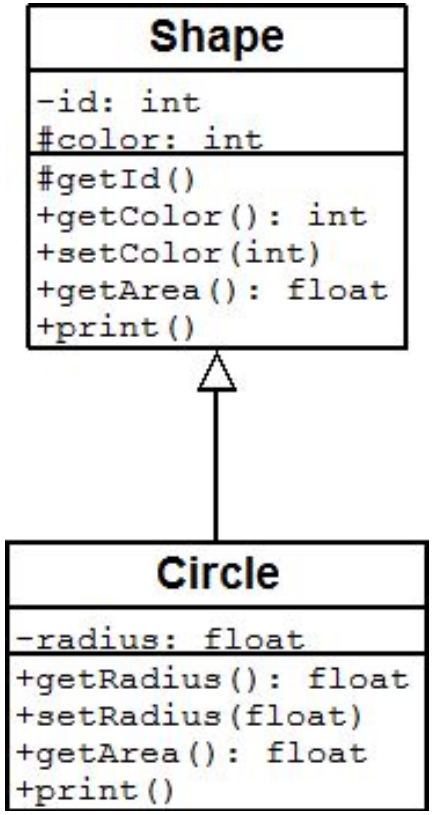
- Une méthode d'une classe enfant **ne peut pas accéder** directement aux membres **privés** de la classe parent.

- Une méthode d'une classe enfant **peut accéder** directement aux membres **publics** ou **protégés** de la classe parent.

Exemples

CircleTest.cpp

```
... // all required lines

int main()
{
        Circle c;
        cout << "My color is " << c.getColor() << endl;
        return 0;
}
```

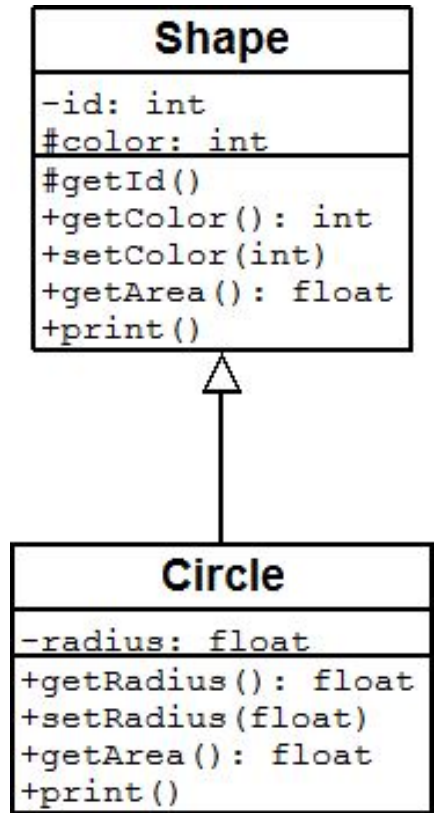


✔ Correct, getColor() est publique

Exemples

CircleTest.cpp

```
... // all required lines

void Circle::print()
{
        cout << "My color is " << getColor() << endl;
}
```

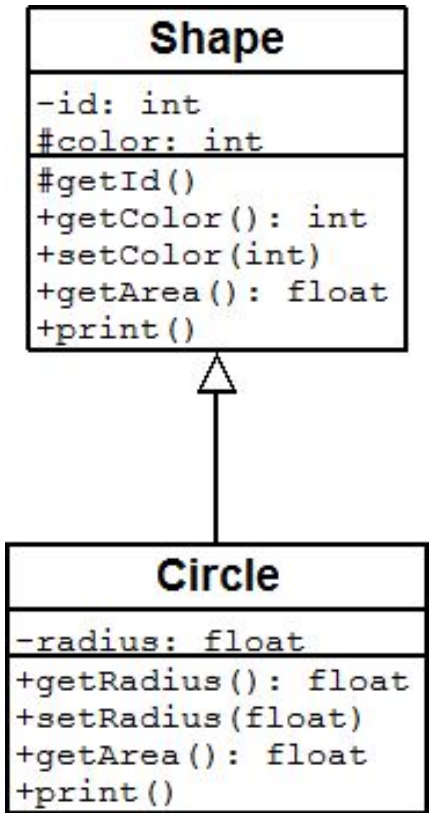


✔ Correct, getColor() est publique

Exemples

CircleTest.cpp

```
... // all required lines

int main()
{
        Circle c;
        cout << "My color is " << c.color << endl;
        return 0;
}
```

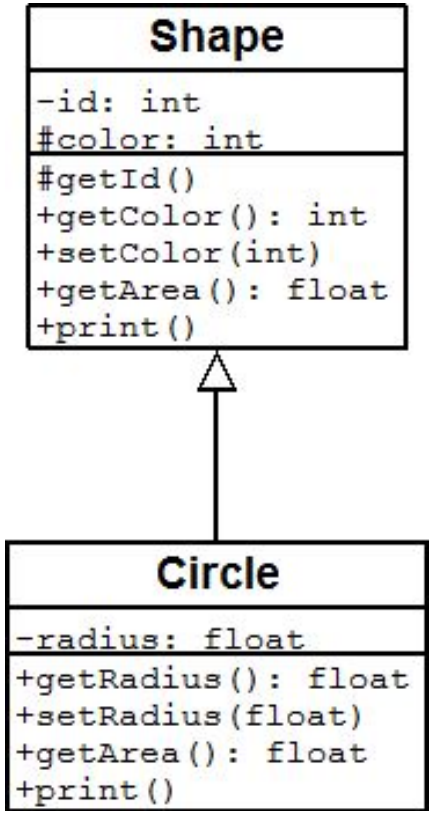


✗ Incorrect: color est protégé

Exemples

CircleTest.cpp

```
... // all required lines

void Circle::print()
{
        cout << "My color is " << color << endl;
}
```



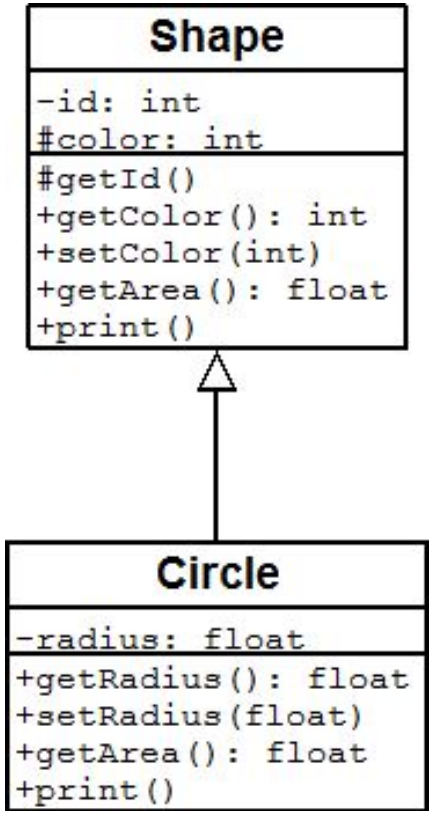
✔ Correct, color est protégé

Exemples

CircleTest.cpp

```
... // all required lines

int main()
{
        Circle c;
        cout << "My id is " << c.id << endl;
        return 0;
}
```

✗ Incorrect: id est privé



Exemples

CircleTest.cpp

```
... // all required lines

void Circle::print()
{
        cout << "My id is " << id << endl;
}
```

✗ Incorrect: id est privé