

Абстрактни класове и полиморфизъм

Що е полиморфизъм?

- Общо: Свойството на даден елемент да приема повече от една форма
- Видове полиморфизъм
 - претоварване на функции: различна реализация в зависимост от типовете на параметрите
`Rational operator+(Rational x, Rational y);`
 - параметричен типов полиморфизъм: една и съща реализация независимо от типовете
`template <typename T> swap(T& x, T& y);`
 - **подтипов полиморфизъм**: една и съща сигнатура за всички подтипове на даден тип
`void Player::print() const;`

Защо ни е полиморфизъм?

- Защо ни е цикъл?
- Защо ни е функция от по-висок ред?
- Защо ни е шаблон?
- Искаме да можем да обработваме много сходни форми по унифициран начин

Що е интерфейс?

- Множество от операции, които поддържа даден тип
 - не включва физическото представяне на типа
 - не включва реализацията на операциите
 - включва имената на операциите
 - включва брой и типове на параметрите
- Ако няколко класа имат общ интерфейс, с тях може да се работи унифицирано
 - но затова всички операции от интерфейса трябва да са виртуални, т.е. с динамично свързване

Примери за интерфейси

- Интерфейс на автомобил
 - волан
 - педали
 - скоростен лост
 - фарове, стопове, мигачи
 - лостове за управление на светлините
 - клаксон
- Учим се да шофираме конкретна кола, а всъщност научаваме да шофираме която и да е кола!

Примери за интерфейси

- Интерфейс на Player
 - наследници: Hero, SuperHero, Bot, Boss
- `void print() const;`
 - всеки наследник предоставя нова реализация, която надгражда предната
- селектори и мутатори за `points`, `name`
 - използва се първоначалната реализация в Player

Примери за интерфейси

- Интерфейс за задача (Task)
 - QuickTask — бърза задача от една единица време
 - SimpleTask — проста (атомарна) задача за няколко единици време
 - изхвърляне на боклука
 - RepeatTask — задача от няколко повтарящи се задачи
 - 20 лицеви опори
 - ComplexTask — сложна (съставна) задача от други задачи
 - вземане на курс във ФМИ
- `void print() const;`
 - отпечатва информация за задачата
- `int time() const;`
 - отпечатва брой единици време за изпълнение на задачата
- селектор и мутатор за `progress`
 - продължава изпълнението на задачата

Примери за интерфейси

- Интерфейс на абстрактен тип данни Stack
 - Stack — последователна реализация чрез масив
 - LinkedStack — свързана реализация
 - `bool push(T const&);`
 - `bool pop(T &);`
 - `bool empty() const;`
- всеки наследник има собствена реализация на член-данните и член-функциите

Видове наследяване

- Наследяване на реализация (имплементация)
 - наследниците споделят член-данни
 - наследниците споделят (частично) реализация на член-функции
 - пример: Player
- Наследяване на интерфейси
 - наследниците споделят сигнатури на член-функции, но предлагат различна реализация
 - пример: Stack
- Смесено наследяване
 - пример: Task

Абстрактни класове

- Наследяването на интерфейси в C++ се реализира чрез **чисти виртуални функции**
- **virtual** <сигнатура> = 0;
- Освобождава ни от задължението да представим дефиниция за тази член-функция
- Клас в който има поне една чиста виртуална функция се нарича **абстрактен**
- **Не можем да създаваме обекти от абстрактен клас!**
- Абстрактен клас, който няма член-данни и всички член-функции са виртуални чисти се нарича **интерфейс**

Абстрактни класове

- Ако наследник на абстрактен клас не реализира някоя чиста виртуална функция, то той също остава абстрактен
- Абстрактните класове могат да имат конструктори и деструктори
 - но те винаги се извикват косвено, от наследник
- Можем да имаме указатели и псевдоними към абстрактни класове
- Абстрактни класове не могат да са
 - параметри на шаблон
 - тип на връщан резултат

Хетерогенни контейнери

- Контейнери, които съдържат обекти от различен тип
- Реализират се чрез използване на указател към полиморфен тип
 - защо указател, а не директно обект?
 - защо указател, а не псевдоним?
- Над хетерогенните контейнери могат да се изпълняват масови операции от общия интерфейс
 - `print()`

Йерархия от задачи

- Да се реализира йерархията от задачи
 - кой ще е общият интерфейс Task?
 - могат ли да се извлекат общи член-данни и реализация в базов абстрактен клас? кои са те?
 - какви член-данни ще има QuickTask?
 - какви член-данни ще има SimpleTask?
 - как RepeatTask ще помни коя задача ще повтаря?
 - може ли RepeatTask да преизползва функционалност от някой от другите класове?
 - как RepeatTask ще възстановява повтаряната задача, когато трябва да я започне отначало?

Обектно-ориентиран дизайн

- Един от подходите за организация на програмен код
- Описва структурата на класовете и връзките между тях
 - какви класове ще има и какви концепции от проблемната област ще описват
 - какви атрибути и операции ще поддържат
 - може ли да се извлече общ интерфейс
 - може ли да се извлече обща функционалност
 - може ли да се преизползва функционалност чрез наследяване

Шаблони за дизайн

- Често срещани и изпитани рецепти за решаване на задачи от обектно-ориентиран дизайн
- Всеки шаблон адресира конкретно изискване или задача
- Използването на шаблони води до по-гъвкава и разширяема програма
- Прекомерното използване на шаблони води до тежка и трудна за поддържане програма
- Добрите готвачи са не тези, които могат да следват рецепти, а тези, които ги използват в правилния момент

Прототип (Prototype)

- Динамично създаване на копие на даден обект, без да е предварително известен неговият тип
- ```
class Cloneable {
public:
 virtual Cloneable* clone() const = 0;
};
```
- ```
class A : public Cloneable {  
    Cloneable* clone() const { return new A(*this); }
```
- ```
Cloneable *prototype, *current;
```
- ```
void reset() {  
    delete current;  
    current = prototype->clone();  
}
```


Влагане на повторения

- Тъй като RepeatedTask е Task, можем да повтаряме повтаряема задача!
- `new RepeatedTask(„level1“, 5,
new RepeatedTask(„level2“, 10,
new RepeatedTask(„level3“, 15,
new SimpleTask(„end“, 3)));`

ComplexTask

- Искаме да направим задача, която се състои от много други задачи
- Можем ли да преизползваме вече написан клас?

Композиция (Composite)

- Позволява група от обекти да бъдат третирани като един обект
- `class Node { ... void print() const; ... };`
- `class Leaf : public Node { ... };`
- `class Composite : public Node {
 ...
 void addNode(Node*, ...);
 bool removeNode(...);
 Node* getNode(...);
 ...
};`

Хетерогенни дървовидни структури

- Шаблонът Composite позволява построяването на дървовидни структури с произволна дълбочина

