Абстрактни класове и полиморфизъм

Трифон Трифонов

Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
2018/19 г.

8 май–5 юни 2019 г.

Що е полиморфизъм?

Що е полиморфизъм?

- Свойството на даден елемент да приема повече от една форма

Що е полиморфизъм?

- Свойството на даден елемент да приема повече от една форма
- Видове полиморфизъм

Що е полиморфизъм?

- 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);
```

Що е полиморфизъм?

- 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);
    ```

Що е полиморфизъм?

- 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 полиморфизъм?

- Защо ни е цикъл?
- Защо ни е функция от по-висок ред?

Защо ни е полиморфизъм?

- Защо ни е цикъл?
- Защо ни е функция от по-висок ред?
- Защо ни е шаблон?

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

Що е интерфейс?

- Множество от операции, които поддържа даден тип
  - не включва физическото представяне на типа

Що е интерфейс?

- Множество от операции, които поддържа даден тип
  - не включва физическото представяне на типа
  - не включва реализацията на операциите

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

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

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

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

Примери за интерфейси

- Интерфейс на автомобил
  - волан

Примери за интерфейси

- Интерфейс на автомобил
  - волан
  - педали

Примери за интерфейси

- Интерфейс на автомобил
  - волан
  - педали
  - скоростен лост

Примери за интерфейси

- Интерфейс на автомобил
  - волан
  - педали
  - скоростен лост
  - светлинно табло

Примери за интерфейси

- Интерфейс на автомобил
  - волан
  - педали
  - скоростен лост
  - светлинно табло
  - лостове за управление на светлините

Примери за интерфейси

- Интерфейс на автомобил
  - волан
  - педали
  - скоростен лост
  - светлинно табло
  - лостове за управление на светлините
  - клаксон

Примери за интерфейси

- Интерфейс на автомобил
  - волан
  - педали
  - скоростен лост
  - светлинно табло
  - 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`

Примери за интерфейси

- Интерфейс на `Player`
  - наследници: `Hero`, `SuperHero`, `Bot`, `Boss`

Примери за интерфейси

- Интерфейс на `Player`
  - наследници: `Hero`, `SuperHero`, `Bot`, `Boss`
- `void print() const;`

Примери за интерфейси

- Интерфейс на `Player`
  - наследници: `Hero`, `SuperHero`, `Bot`, `Boss`
- `void print() const;`
  - всеки наследник предоставя нова реализация, която надгражда предната

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 `score`, `name`

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 `score`, `name`
  - използва се първоначалната реализация в `Player`

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив
  - `ResizingStack` — разширяваща се реализация чрез динамичен масив

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив
  - `ResizingStack` — разширяваща се реализация чрез динамичен масив
  - `LinkedStack` — свързана реализация

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив
  - `ResizingStack` — разширяваща се реализация чрез динамичен масив
  - `LinkedStack` — свързана реализация
  - `bool push(T const&);`

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив
  - `ResizingStack` — разширяваща се реализация чрез динамичен масив
  - `LinkedStack` — свързана реализация
  - `bool push(T const&);`
  - `bool pop(T &);`

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив
  - `ResizingStack` — разширяваща се реализация чрез динамичен масив
  - `LinkedStack` — свързана реализация
  - `bool push(T const&);`
  - `bool pop(T &);`
  - `bool empty() const;`

Йерархия от стекове

- Интерфейс на абстрактен тип данни `Stack`
  - `Stack` — последователна реализация чрез масив
  - `ResizingStack` — разширяваща се реализация чрез динамичен 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)

Видове наследяване

- Наследяване на реализация (имплементация)
  - наследниците споделят член-данни

Видове наследяване

- Наследяване на реализация (имплементация)
  - наследниците споделят член-данни
  - наследниците споделят (частично) реализация на член-функции

Видове наследяване

- Наследяване на реализация (имплементация)
  - наследниците споделят член-данни
  - наследниците споделят (частично) реализация на член-функции
  - **Пример:** `Player`

Видове наследяване

- Наследяване на реализация (имплементация)
  - наследниците споделят член-данни
  - наследниците споделят (частично) реализация на член-функции
  - **Пример:** `Player`
- Наследяване на интерфейси

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

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`

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

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
  - споделят се някои член данни

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
  - споделят се някои член данни
  - някои функции споделят реализация

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
  - споделят се някои член данни
  - някои функции споделят реализация
  - други функции споделят само сигнатура, но предлагат различна реализация

Абстрактни класове

- Наследяването на интерфейси в C++ се реализира чрез **чисти виртуални функции**

Абстрактни класове

- Наследяването на интерфейси в C++ се реализира чрез **чисти виртуални функции**
- `virtual` <сигнатура> `= 0;`

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

Абстрактни класове

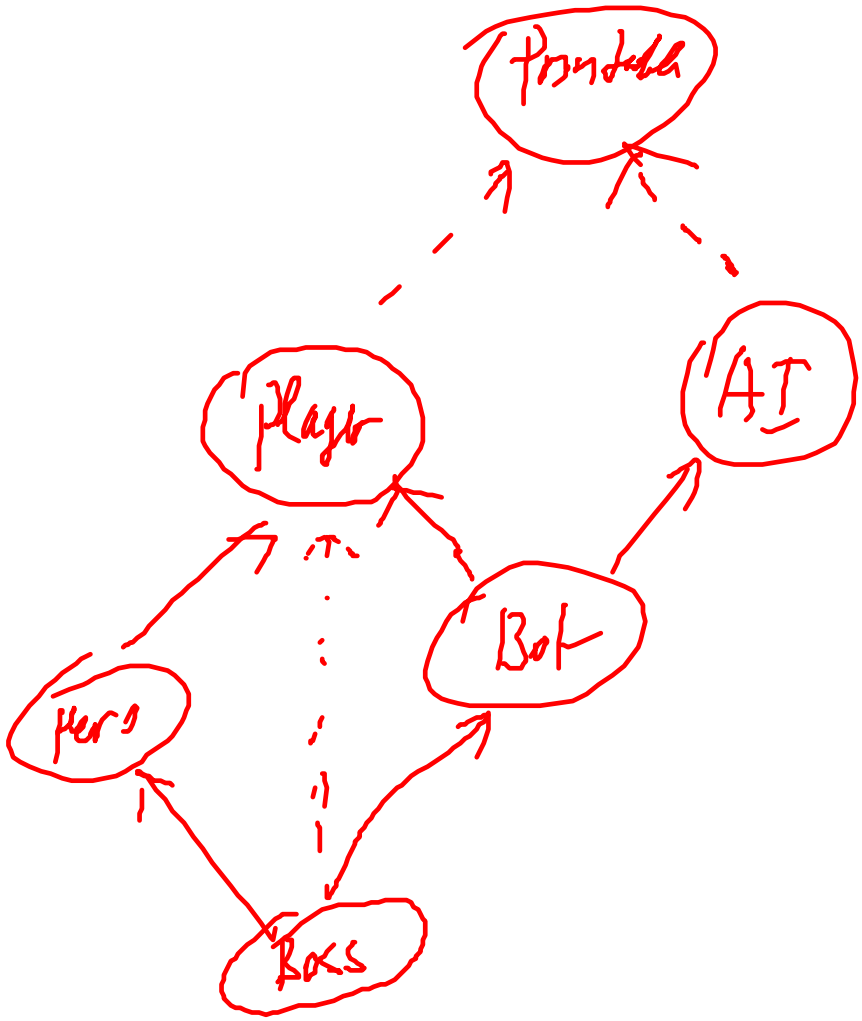
- Наследяването на интерфейси в C++ се реализира чрез **чисти виртуални функции**
- `virtual` <сигнатура> `= 0;`
- Освобождава ни от задължението да представим дефиниция за тази член-функция
- Клас в който има поне една чиста виртуална функция се нарича **абстрактен**

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!

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**


Player
AI
Bot
Hero
Boss

Особености на абстрактните класове

- Ако наследник на абстрактен клас не реализира някоя чиста виртуална функция, то той също остава абстрактен

Особености на абстрактните 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

Особености на абстрактните 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 производен клас

Особености на абстрактните 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 производен клас
- Можем да имаме указатели и препратки към абстрактни класове

Особености на абстрактните 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 производен клас
- Можем да имаме указатели и препратки към абстрактни класове
- Абстрактни класове не могат да са

Особености на абстрактните 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 производен клас
- Можем да имаме указатели и препратки към абстрактни класове
- Абстрактни класове не могат да са
  - тип на член-данни (защо?)

Особености на абстрактните 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 производен клас
- Можем да имаме указатели и препратки към абстрактни класове
- Абстрактни класове не могат да са
  - тип на член-данни (защо?)
  - тип на връщан резултат (защо?)

Хетерогенни контейнери

- Контейнери, които съдържат обекти от различен тип

Хетерогенни контейнери

- Контейнери, които съдържат обекти от различен тип
- Реализират се чрез използване на указател към полиморфен тип

Хетерогенни контейнери

- Контейнери, които съдържат обекти от различен тип
- Реализират се чрез използване на указател към полиморфен тип
  - защо указател, а не директно обект?

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 препратка?

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 препратка?
- Над хетерогенните контейнери могат да се изпълняват масови операции от общия интерфейс

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 препратка?
- Над хетерогенните контейнери могат да се изпълняват масови операции от общия интерфейс
  - **Пример:** `print()`

Йерархия от задачи

- Интерфейс за задача (`Task`)

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи
    - **Пример:** вземане на изпит във ФМИ

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи
    - **Пример:** вземане на изпит във ФМИ
- име на задачата

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи
    - **Пример:** вземане на изпит във ФМИ
- име на задачата
- извеждане на информация за задачата

Йерархия от задачи

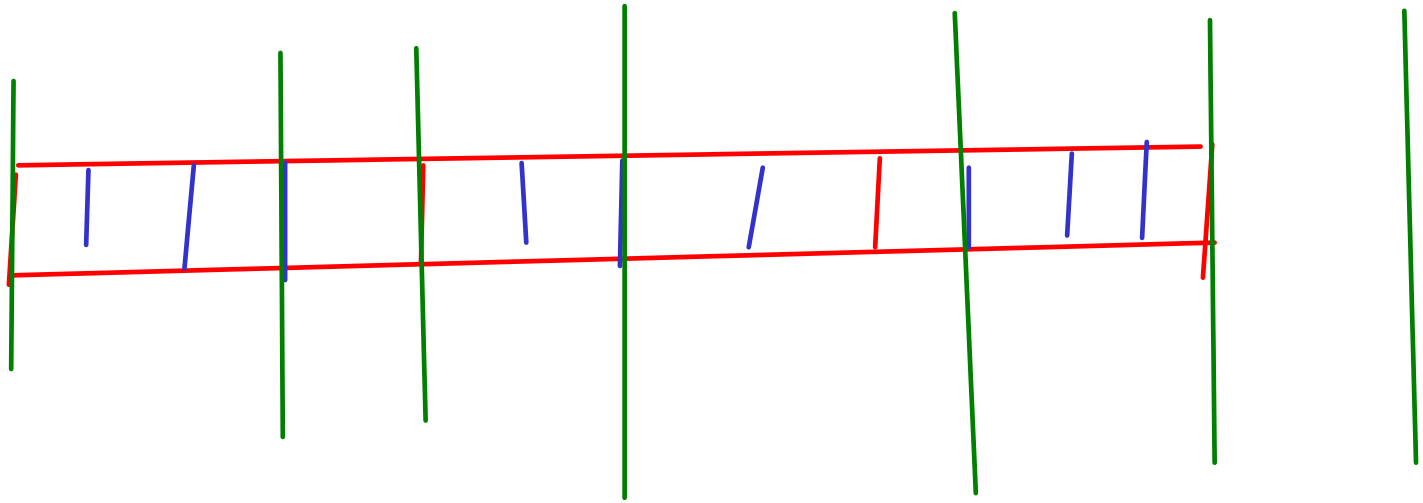
- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи
    - **Пример:** вземане на изпит във ФМИ
- име на задачата
- извеждане на информация за задачата
- селектор за брой единици време за изпълнение на задачата

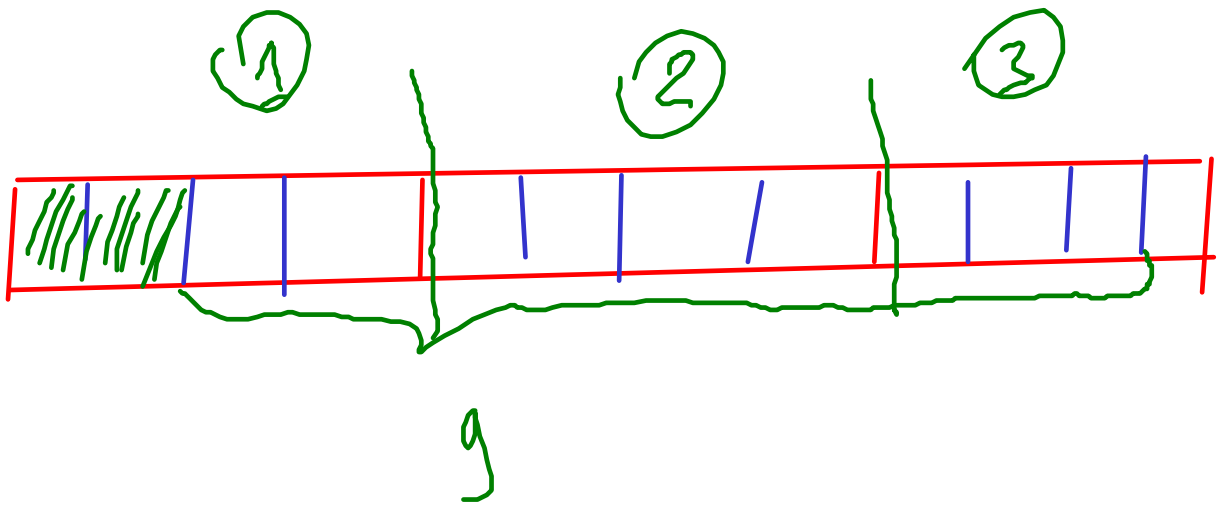
Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи
    - **Пример:** вземане на изпит във ФМИ
- име на задачата
- извеждане на информация за задачата
- селектор за брой единици време за изпълнение на задачата
- селектор за прогрес на задачата

Йерархия от задачи

- Интерфейс за задача (`Task`)
  - `QuickTask` — бърза задача от една единица време
  - `SimpleTask` — проста (атомарна) задача за няколко единици време
    - **Пример:** изхвърляне на боклука
  - `RepeatTask` — задача от няколко повтарящи се задачи
    - **Пример:** 20 лицеви опори
  - `ComplexTask` — сложна (съставна) задача от други задачи
    - **Пример:** вземане на изпит във ФМИ
- име на задачата
- извеждане на информация за задачата
- селектор за брой единици време за изпълнение на задачата
- селектор за прогрес на задачата
- мутатор за прогрес на задачата (работа по задачата)




1
2
3
9

Реализация на йерархията

- кой ще е общият интерфейс `Task`?

Реализация на йерархията

- кой ще е общият интерфейс `Task`?
- могат ли да се извлекат общи член-данни и реализация в базов абстрактен клас? кои са те?

Реализация на йерархията

- 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`?

Реализация на йерархията

- 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`?

Реализация на йерархията

- 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?

Реализация на йерархията

- 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?

Реализация на йерархията

- 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

Обектно-ориентиран дизайн

- Един от подходите за организация на програмен код
- Описва структурата на класовете и връзките между тях

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

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

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

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

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

- Изпитани рецепти за решаване на често срещани задачи от обектно-ориентиран дизайн

Шаблони за дизайн

- Изпитани рецепти за решаване на често срещани задачи от обектно-ориентиран дизайн
- Всеки шаблон адресира конкретно изискване или задача

Шаблони за дизайн

- Изпитани рецепти за решаване на често срещани задачи от обектно-ориентиран дизайн
- Всеки шаблон адресира конкретно изискване или задача
- Използването на шаблони води до по-гъвкава и разширяема програма

Шаблони за дизайн

- Изпитани рецепти за решаване на често срещани 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

Шаблони за дизайн

- Изпитани рецепти за решаване на често срещани 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 по правилния начин

Прототип (Prototype)

Динамично създаване на копие на даден обект, без да е предварително известен неговият тип.

```
class Cloneable {
public:
  virtual Cloneable* clone() const = 0;
};
```

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); }
};
```

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;
```

Прототип (Prototype)

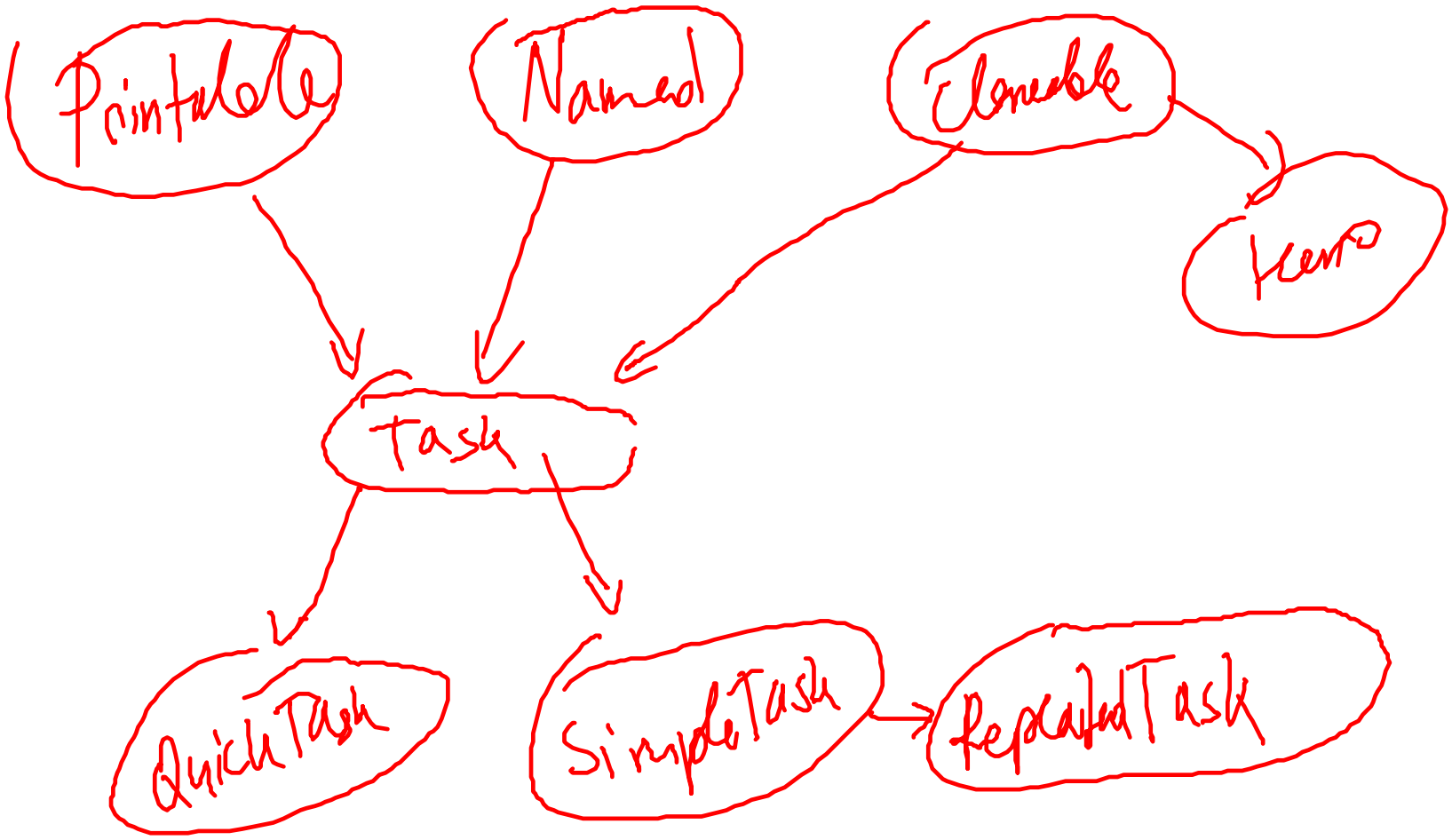
Динамично създаване на копие на даден обект, без да е предварително известен неговият тип.

```
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();
}
```


Printable
Named
Cloneable
Task
QuickTask
SimpleTask
RepeatedTask

Влагане на повторения

Тъй като `RepeatedTask` е `Task`, можем да повтаряме повтаряема задача!

Влагане на повторения

Тъй като RepeatedTask е Task, можем да повтаряме повтаряема задача!

```
new RepeatedTask("бакалавър",
  8, new RepeatedTask("семестър",
    4, new RepeatedTask("курс",
      15, new SimpleTask("лекция", 3))));
```

Влагане на повторения

Тъй като RepeatedTask е Task, можем да повтаряме повтаряема задача!

```
new RepeatedTask("бакалавър",
  8, new RepeatedTask("семестър",
    4, new RepeatedTask("курс",
      15, new SimpleTask("лекция", 3))));
```

Вариант на шаблона "верига отговорности" (Chain of Responsibility)