Poznan University of Technology
Object Oriented Programming
Constructors

Jakub Piotr Hamerliński, M.Eng.

https://www.linkedin.com/in/hamerlinski

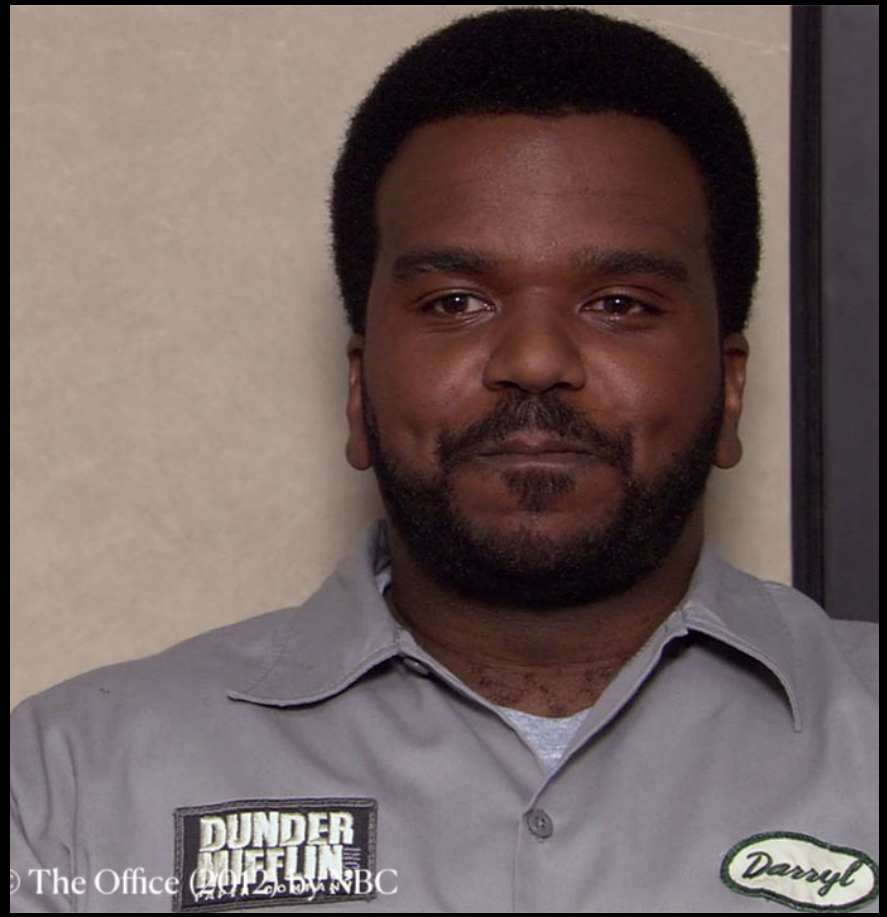
https://github.com/hamerlinski

Constructors

Agenda

01. Introduction
02. Types
03. Delegation
04. Code inside constructors
05. Destructor
06. Task

Bookstore employee: *Well, if you read a lot, you should check out our e-readers. They're really neat.*
Darryl: *Whoa, I work at a paper company, those things terrify me. They could put us out of business, you know? I heard those machines hold, like, ten books at once.*
Bookstore employee: *Actually, it's ten thousand.*
Darryl: *Holy [bleep]. What? Let me see that. It's so dark. Like a croissant.*



Constructors

Constructors

Introduction

It’s an entry point to a new object.
It accepts some arguments, does something with them, and prepare the object to perform their duties.
Properly constructed class should have many constructors and only a few methods.

> **Number of constructors** > **Number of methods**

Constructors

Examples

```
Money defaultCash;
Money floatCash(140.15f);
Money stringCash("128.12");
Money intCash(144);
```

Constructors

Types

Constructor's main task is to initialize encapsulated properties, using the provided arguments.
One primary constructor with initialization.
Many secondaries constructors, which call the primary one.
This approach reduces complexity and helps to avoid duplication.

Constructors

Delegation

```
class Money {
 private:
  double amount;
 public:
  double Balance() { return this->amount; };
  Money() { this->amount = 0.0; };
  Money(double mny) { this->amount = mny; };
  Money(float mny) : Money(static_cast<double>(mny)) {};
  Money(int mny) : Money(static_cast<double>(mny)) {};
  Money(std::string mny) : Money(std::stod(mny)) {};
};
```

Constructors

Delegation

```
int main() {
  Money defaultCash;
  Money doubleCash(69.69);
  Money floatCash(360.01f);
  Money stringCash("21.37");
  Money intCash(420);
  std::cout << defaultCash.Balance() << std::endl;
  std::cout << doubleCash.Balance() << std::endl;
  std::cout << floatCash.Balance() << std::endl;
  std::cout << stringCash.Balance() << std::endl;
  std::cout << intCash.Balance() << std::endl;
  std::cout << "typeid of stringCash.Balance() -> " << typeid(stringCash.Balance()).name();
  return 0;
}
// output:
// 0
// 69.69
// 360.01
// 21.37
// 420
// typeid of stringCash.Balance() -> d
```

Constructors

Code inside constructors

Any computations done inside a constructor is a bad practice, because they can create side effects that are not requested by the object owner.

> Constructs must be code-free.

Constructors

Code inside constructors - example

```
class Name {
  virtual std::string FirstName() = 0;
};
class EnglishName : public Name {
 private:
  std::string text;
 public:
  EnglishName(std::string txt) { this->text = txt; }
 public:
  std::string FirstName() {
    return text.substr(0, text.find(" "));
  }
};
```

Constructors

Destructor

A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete. A destructor has the same name as the class, preceded by a tilde (~). For example, the destructor for class String is declared: ~String().

If you don’t define a destructor, the compiler provides a default one; for many classes this is sufficient. You only need to define a custom destructor when the class stores handles to system resources that need to be released, or pointers that own the memory they point to.

Constructors

Destructor example

```
#include <string>
class String {
 public:
  String(char *ch);                 // Declare constructor
  ~String();                        // and destructor.
 private:
  char *_text;
  size_t sizeOfText;
};
String::String(char *ch) {          // Define the constructor.
  sizeOfText = strlen(ch) + 1;
  _text = new char[sizeOfText];     // Dynamically allocate the correct amount of memory.
  if (_text)
    strcpy(_text, ch);              // If the allocation succeeds, copy the initialization string.
}
String::~String() {                 // Define the destructor.
  delete[] _text;                   // Deallocate the memory that was previously reserved for this string.
}
int main() {
  String str("The point is there ain't no point.");
}
```

Task

Constructors

Task

Task

01. Write a program which will showcase delegation of the constructors and the flexibility it provides.

Constructors

Sources

01. Destructors (C++)

Thank you

Feel free to reach me via LinkedIn

Fin