05 - Constructors and Destructors

January 23, 2023

COMP2404

Darryl Hill

Contents

Default Arguments

Argument is a parameter value

- A ***default*** argument is a default parameter value
- similar to default arguments in Python
- no default arguments in Java 8

Properties of default arguments:

- specified in the function prototype
  - NOT in the implementation - compiler will complain
- may have as many default arguments as we like, HOWEVER
- default arguments must be the rightmost arguments of a parameter list

programming example <p1>

Default Arguments

Default arguments can be used in

- global functions
- member functions
- constructors

Must be careful of the function signature!

```
void fun1 (int = 0);
void fun1 ();
```

If a call is made to `fun1()`, which of these is called?

Default Member Functions

Every class is provided with 4 member functions:

1) default constructor (ctor)
2) default destructor (dtor)
3) copy constructor (more on this shortly)
4) assignment operator

We may want to override these functions (particularly if the class contains dynamic memory):

- Rule of three - if we need user defined versions of 2, 3, or 4, then we probably need all three.
- Rule of five - modern compilers also include move constructors and move assignment. operators - not covered in this class.
- Rule of two - There may only be two Sith Lords at any given time.

Default Constructors

A default constructor may be:

- A zero argument constructor.
- A constructor with all default arguments.

```
class Date {
     Date(int = 1901, int = 1, int = 1);
     Date();
}

Date today;
```

Beware. Which constructor is called?

- A constructor no arguments and with all default values has the same function signature.

Default Constructors

You are allowed 1 default constructor per class

- If you do not provide ***any*** constructor, a default constructor is provided for you.
- If you provide a constructor (say with 2 arguments) you no longer get a default constructor for free
  - but you can still make one.

Constructors have 1 purpose:

- Initialize the data members.

We should initialize **all data members** in **every constructor**.

- If you do not initialize a member, your code will work ***sometimes***.
- These are the worst types of errors to try to find.

programming example <p2>

Default Constructors

Default constructors are called implicitly when

- an object is declared with no parameters

```
Date d1;
```

- an array of objects is declared

```
Date dates[10];
```

  - the default constructor is called for each element

- When memory for an object is dynamically allocated using the ***new*** operator
  - This is allocated on the **Heap**
  - ***dynamically allocated***

```
Date* d1 = new Date;
```

Destructors

Destructors

- Member function of a class
- One is provided if we do not write it
- Called at the end of the life cycle of an object

One destructor per class

- Has no arguments (even if we write one).
- `delete` will immediately trigger the destructor for ***dynamic memory only***.
- For **statically allocated memory** it is called **implicitly** at the **end of the object life cycle**...thus you should know when it is called.

ctor: `Date()`

dtor: `~Date()` (using a tilde character)

Destructor Usage

Performs cleanup and release resources.

- Release memory, close files, save state, etc.
- We will mainly use it to release dynamically allocated memory.
  - Call `delete` any data members declared using `new`.

You must write the destructor code.

- Compiler has no idea what resources should be released or memory deleted.

Destructor Usage

Destructor called when:

- Locally declared object moves out of scope.
- At the end of the program (for global objects).
- When `delete` is called on a ***dynamically allocated object*** (allocated with `new`)

For multiple objects, destructors are usually called in the reverse order of the constructors (but not always).

programming example <p3>

Destructors

If we call `exit()`, program terminates immediately.

- Global variables are destroyed, local variables are not.

If `abort()` is called, no destructors are called.

- For unrecoverable failure.

Copy Constructors

A member function of a class

Takes a single argument only

- reference to an object of the same class

  `Date(Date&)` or

  `Date(const Date&)` //const means we cannot modify it
- Anything other than above and the compiler ***will not*** recognize it.
- Will ***not*** take an object.
- Will ***not*** take a pointer.

May be called explicitly – one example: `Date d2(d1);`
Can also be called implicitly (in unexpected ways).

Copy Constructors

A copy constructor is provided by default, HOWEVER

- performs member-wise assignment.
- If there is a pointer to dynamic memory, only the address is copied.
  - So now our two objects will have pointers the same data member.
  - Called a ***shallow copy***, as opposed to a ***deep copy***.

Conversion constructor has a similar job, but takes a referenct to an object of a different class.

- More on this later.

Copy Constructors

Usage:

- Make a copy of an existing object.
- Initialize values of an object with values of another object.
  - We can access the private members for this.
  - This is different from Java.

Declaration: `Student matilda;`

- **Calls (default) constructor**

Initialization: `Student matilda = bertha;`

- **Calls copy constructor**

assignment: `matilda = bertha`

- **Calls assignment operator** – here `matilda` is not a new object - already exists.

Copy Constructors

Copy constructors can be called explicitly:

- on declaration

```
Date d2(d1);
```

Copy constructors can be called implicitly:

- when an object is ***passed*** by value as a function parameter
- on **initialization**

```
Date d1;
Date d2 = d1;
```

- this is **NOT** the same as **assignment**

coding example <p4>

Conversion Constructors

Similar to a copy constructor
A conversion constructor:

- is a member function of a class (class A)
- takes a parameter of a different data type (type B)

A conversion constructor is called when

- passed in as a constructor argument - if this copy constructor exists

```
DateTime(Date&);
```

- upon initialization

```
Date d1;
DateTime dt = d1;
```

- using pass (or return) by value.

Conversion Constructors

A conversion constructor uses the members of one type to initialize the members of another type.

- These should logically be related types.

The use of a conversion constructor can be disabled.

- Use the `explicit` keyword.
- Disables implicit uses.
- Explicit uses are still permitted.

programming example <p5>