Initialization & Finalization

John Nelson and
Matthew Gallagher

Presentation Topics

- Why we chose the topic
- Overview/Purpose
- Languages used In
- History
- Examples in languages
- Challenges
- Pros and Cons
- Short Quiz

Why We Chose Initialization and Finalization

- Our Studied at Montana Tech have had us use these constructs a lot over the years.
- Occurs in Java and C++ languages.
- Always knew we had to initialize object and variables. Which pique our interest in learn the about why it is important.

Overview/Purpose

- Initialization and finalization are a major construct of most imperative languages.

- Initialization's purpose is to facilitate assign of space and values to objects and variables in memory.

- Finalization's purpose is to facilitate the deallocation of objects and variables from memory when they are no longer used.

Languages Used In

- Java
- C++
- Fortran
- Common Lisp
- Python
- Simula

History of Initialization

The beginning of initialization started with the use of variables in formulas introduced by FORTRAN in 1954. This was the first imperative programming language to be released. In OOL, initialization is used also when constructing an object.

- Simula (1967)-
  - First object-oriented programming language introducing classes.
- SmallTalk (1980) -
  - Classes are objects, and instances are another object. SmallTalk uses messages to access behaviors of a class from the class meta-data.
- C++ (1983) -
  - Explicit definition of value types.
  - Object can be created implicitly.

History of Initialization (Cont.)

- Python (1991) -
  - Implicit definition of value types.
  - Uses special constructor for classes to intialize an object.
- Java (1995) -
  - This is the first language to allow for the creation of a default constructor without the need to type it.

History of Finalization

- Common LISP (1958) -
  - First to implement garbage collection
- C++ (1983) -
  - Introduced destructors for dealing with "garbage"
  - Later versions include smart pointers in the 1990's
- Java (1995) -
  - Automated garbage collection

Simula Initialization

```
Class Rectangle (Width, Height); Real Width, Height;
                              ! Class with two parameters;
 Begin
    Real Area, Perimeter;   ! Attributes;

    Procedure Update;       ! Methods (Can be Virtual);
    Begin
      Area := Width * Height;
      Perimeter := 2*(Width + Height)
    End of Update;

    Boolean Procedure IsSquare;
      IsSquare := Width=Height;

    Update;                 ! Life of rectangle started at creation;
    OutText("Rectangle created: "); OutFix(Width,2,6);
    OutFix(Height,2,6); OutImage
 End of Rectangle;
```

http://staff.um.edu.mt/jskl1/talk.html

Java Initialization

```java
public class Rectangle {
    int Width = 0;
    int Height = 0;

    public Rectangle(int W, int H) {
        this.Width = W;
        this.Height = H;
    }
    public int area() {
        return Width * Height;
    }
}

public class Demo {
    public static void main(String[] args) {
        int W = 6;
        int H = 4;

        Rectangle Mine = new Rectangle(W,H);
        int A = Mine.area();
        System.out.println(A);

    }

}
```

Common Lisp Finalization

```
(garbage-collect)
     ⇒ ((conses 16 49126 8058) (symbols 48 14607 0)
                (strings 32 2942 2607)
                (string-bytes 1 78607) (vectors 16 7247)
                (vector-slots 8 341609 29474) (floats 8 71 102)
                (intervals 56 27 26) (buffers 944 8)
                (heap 1024 11715 2678))
```

http://www.gnu.org/software/emacs/manual/html_node/elisp/Garbage-Collection.html

C++ Finalization

```
// order_of_destruction.cpp
#include <cstdio>

struct A1       { virtual ~A1() { printf("A1 dtor\n"); } };
struct A2 : A1 { virtual ~A2() { printf("A2 dtor\n"); } };
struct A3 : A2 { virtual ~A3() { printf("A3 dtor\n"); } };

struct B1       { ~B1() { printf("B1 dtor\n"); } };
struct B2 : B1 { ~B2() { printf("B2 dtor\n"); } };
struct B3 : B2 { ~B3() { printf("B3 dtor\n"); } };

int main() {
    A1 * a = new A3;
    delete a;
    printf("\n");

    B1 * b = new B3;
    delete b;
    printf("\n");

    B3 * b2 = new B3;
    delete b2;
}
```

https://docs.microsoft.com/en-us/cpp/cpp/destructors-cpp?view=msvc-170

Challenges

- Constructor naming
  - Certain languages allow multiple constructors for a class. To differentiate, a language will call parameters to interpret the correct constructor.
  - SmallTalk and Eiffel use a number to system to differentiate constructors.

```
Day()
{
    int year = -1;
    int month = -1;
    int day = -1;
}

Day(int year, int month, int day)
{
    year = year;
    month = month;
    day = day;
}
```

Challenges (Cont.)

- Reference vs. Values
  - Languages such as C++ and Ada allow variables to be the value of an object
  - SmallTalk, Ruby, Python, and Java use variables to refer an object
  - During initialization in C++, you can initiate an object without specifying a constructor call. This makes it possible to use an object variable before it has a value
  - In C#, you must explicitly call the correct constructor for object creation

Pros of Initialization & Finalization

- Initializing objects and variables space in memory ensures fast performance of a program by only using up the required memory

- The pro of finalization is to free up memory that is no longer being used by an object.

Cons of Initialization & Finalization

- Each language has their own way of using finalization process on objects.
- C++ requires the user to have a higher knowledge of memory management.
- While Java uses an automated garbage collection system to remove unreferenced objects.

Remarks

It is safe to say without initialization and finalization, object-oriented languages would not be able to operate the way they do today.

Initialization has come a long way from its humbled beginnings. The use of variables and objects have become a customary feature in programming languages.

Finalization made memory management efficient for programmers.

Questions ?

Reference

- Corob-Msft. (2021, August 3). *Destructors (C++)*. Microsoft Docs. Retrieved November 19, 2021, from https://docs.microsoft.com/en-us/cpp/cpp/destructors-cpp?view=msvc-170.
- cs.cmu.edu. (n.d.). *19.6. By-Position Constructor Functions*. Common Lisp the Language, 2nd Edition. Retrieved November 19, 2021, from https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node1.html.
- Danny, V. (2019, October 24). *Tutorial oop(iii): Constructors and Destructors*. The Delocalized Physicist. Retrieved November 19, 2021, from https://dannyvanpoucke.be/oop-fortran-tut4-en/.
- DeBrie, A. A. (2021, May 3). *Python garbage collection: What it is and how it works*. Stackify. Retrieved November 19, 2021, from https://stackify.com/python-garbage-collection/.
- Furuknap, B. (2015, December 4). *How to: Object-oriented programming - atlantic.net*. Atlantic.Net Blog. Retrieved November 19, 2021, from https://www.atlantic.net/vps-hosting/how-to-object-oriented-programming-classes-objects/.
- Singh, C., says, P. J., Jakhar, P., & says, C. S. (2018, March 10). *Python constructors - default and parameterized*. beginnersbook.com. Retrieved November 19, 2021, from https://beginnersbook.com/2018/03/python-constructors-default-and-parameterized/.
- Sklenar, J. (1997, December 5). *INTRODUCTION TO OOP IN SIMULA*. Introduction to Simula. Retrieved November 19, 2021, from http://staff.um.edu.mt/jskl1/talk.html.
- Tutorialspoint. (n.d.). *https://www.tutorialspoint.com/java/lang/object_finalize.htm*. Java.lang.Object.finalize() method. Retrieved November 19, 2021, from https://www.tutorialspoint.com/java/lang/object_finalize.htm.
- The User's Guide - 9 SEP 1996. (n.d.). *5.1 About storage in Lisp*. 5 Storage Management in Common Lisp. Retrieved November 19, 2021, from http://www.lispworks.com/documentation/lcl50/ug/ug-67.html.