**NAME Recardo House**

**STUDENT ID: UB73367SC082527**

**PROGRAM: Bachelors in Computer Engineering**

COURSE NAME

**Programming Language for Object**

ATLANTIC INTERNATIONAL UNIVERSITY

**August 2021**

**TABLE OF CONTENTS**

**INTRODUCTION ……………………………………….………………………………….……………………1 ­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­**

Introduction Object-Oriented Programming Overview……..…………………...………………………………………………………..1

Benefits of Object-Oriented…………….…………………………………………………………………… ………………………… ….2

Real World Apprehension………………………………………………………………..……………………….…….…………….……..2

Stability of Design .........................................................................................................................................................................2

Reusability….…….………………………………………………………………………………………………..……………………….…3

Object-Oriented Programming and BETA…………….………………………………………………………..…….………..……….…3

System Development Methods………………………….................................................................................................................6

Logical Versus Physical System Structure……………………………………. ……………………………….……………………….6

BETA……..……………………………………………………………………………………………………….……….……………….…7

Introduction to Basic Concepts………………………………………………….…………….……………………………………………8

Procedural Programming……………………………………………………………………………………...........................................9

Computers…………………………………..……………………………………….………….……………………………………………11

Computer Programming Language……………………………………………………….…………………………………………….…12

Software Systems……………………………………………..…………………………………………………………………..………...12

General Programming Language ………………………..………..…………………………...…………………………………….…...14

The Major Constructs of Structured Programming Languages ………………………………………………………………..………15

Control Structures…………………………………………………………………………………………………………………………...16

**CONCLUSION .….…………………………….........................................................................................................................17**

**BLIOGRAPHY ….………………………………..………………………………………………..………….18**

**1 of 1**

***CHAPTER 1***

***Object-Oriented Programming***

***Overview:***

**INTRODUCTION**

**…………………………………………………………………………**

Programming language often characterized an object-oriented language, a

functional language, procedural language, etc. Term **multi-perspective** (or **multi-**

**paradigm**) l**anguage** is used languages supporting more than one perspectives.

“object-oriented language” used for languages that only support object-orientation.

**Object-oriented language** a programming language support object-oriented

Programming, a **pure object-oriented language** a programming language only

supports object-orientation.

(Lehrmann & Moller, 1993)

 **1 of 2**

**Benefits of object-orientation**

The benefits of object-orientation are considered to be. Three main benefits: real word

apprehension, stability of design and reusability of both designs and implementations.

Let’s consider all three to be important, though perhaps not equally so.

**Real world apprehension**

The object-oriented programming is widely accepted is that object-orientation is close to

own natural perception of real world. (Krogdahl and Olsen, 1986) (translated from

Norwegian) put it this way:

The philosophy underlying object-oriented programming to make programs far as

possible part of reality they are doing to treat. It easier to understand to get an overview

what is described in programs. Human being from outset used and trained in perception

what going in the real world. Thinking of using this programming language, the easier to

write and understand programs.

In (Coad and Yourdon, 1990) is stated in the following way:

 ‘Object-oriented analysis based upon concepts that we first learned in:

 Kindergarten ‘objects and attributes, classes and members, wholes and parts.’

The quotations stress one important aspect of program development understand,

describe and communicate phenomena concepts of application domain. Object-oriented

programming turned to be particularly suited for doing this.

**Stability of design**

The Jackson System Development (JSD) principle method, (Jackson, 1983) reflects the

benefits of object-orientation. The first step system’s development to JSD is to make

*physical model* of real world which the system concerned. The basic for this model

**1 of 3**

forms different functions the system may have. Function changed later, and new

functions may added without changing underlying model.

 The physical model introduced by JSD central to object-orientation. Concept and

techniques used by JSD method develop a physical model, to subsequently implement

on computer are, different from concepts and techniques object-oriented programming.

Object-oriented programming provides natural framework modeling application domain.

**Reusability**

The only problem with software development is to reuse existing software components

when developing new ones. The existing component often similar to one needed for

new system. There important differences make it impossible to reuse existing

component. New component implemented by copying and modifying existing

component, it must be tested again. If error is detached one, it must be correct in both

components.

 The benefits of object-oriented programming language modification. A new

component excremental extension of existing one, preserving relation between two

component. Some object-oriented languages baes on class/subclass mechanism first

appeared in Simula. The Smalltalk was that these language constructs combined

flexibility of Lisp systems.

Without Smalltalk, Ability to create programs by incremental extension the advantage of

object-oriented programming by many programmers. Disadvantage of incremental

modification that library components reflects historic development these components.

Relationship between components dictated by maximal code sharing, conflicting with

modeling requirement.

(Lehrmann & Moller, 1993)

**Object-oriented programming and BETA**

**1 of 4**

**To program is to understand:** The information system development not just writing a

program that does the job. The development of this program revealed in-depth

understanding the application domain; otherwise, information system may not fit into the

organization. Development of systems is important that descriptions application domain

communicated between system specialists and organization.

There more to object-oriented programming than language constructs (true for any

programming language). The framework or semantic just important as the language.

 Other programming based some mathematical theory or model, with a theoretical

basis, object-oriented programming lack theoretical basis. Designing programming

language, benefit of conveying understanding the basic concepts on the language,

model does not have to be formal. Model underlying object-orientation by very natural

informal, part of model been formalized in terms programming languages. The

necessary order to create descriptions that executed on a computer. BETA can see as

formal notation describing parts of application domain to be formalized: BETA is formal

make it a precise meaning when executed on a computer. Programming language

defined in terms of mathematical model, this model not been contracted for BETA, its

useful for purpose consistency checking and formal verification.

 The informal concepts in system development process is important, there a greater

emphasis on concepts that formulated in terms of programming language. The system

development can be executed on a computer. System development, traditionally

organized into *analysis*, *design* and *implementation*, and concepts object-orientation

applied to all activities.

**\* Analysis**. Primary gold of analysis to *understand* application domain. The relationship

concepts and phenomena from application domain be identified and described,

involving the system be developed. Important the concepts phenomena identified can

**1 of 5**

be communicated. The process useful in a high number of informal descriptions, it

difficult to communicate descriptions to non-computer specialists. Informal descriptions

have text mixture, graphics, sometimes incomplete program fragments. Graphic in

system descriptions important, in analysis as be important to graphical notation for the

descriptions.

**\* Design.** This design concerned with constructing precise description can refined into

executable program. Informal concepts developed in analysis activity made into formal

concepts that is described by a programming language like BETA. Object-oriented

program will description of phenomena and concepts from application domain.

Graphical notation useful in design activity. Some programming language like BETA is

textual syntax, and often advantageous using graphic notations for program instead of

textual representation.

Analysis and design similar in descriptions that is meaningful terms of application

domain. Different in respect to their use of informal and formal descriptions.

**\* Implementation**. This *implement* the design description on a computer, i.e. elements

concepts and phenomena from application domain described terms concepts that

executed on computer. These computer concepts do not represent concepts and

phenomena from application domain, i.e. program extended details that are

meaningless in application domain. Basic principles of two levels the same;

programming at different levels.

Designing and implementation are similar that programming language used for

descriptions. They differ that the elements design description be meaning in terms the

**1 of 6**

application domain, Whereas this not the case for implementation description.

**System development methods**

The above analysis, design and implementation give impression these activities ordered

in time (i.e. first analysis, design and then implementation). In some situations activities

may intermixed, the developer not be conscious of them. Many methods of organizing

system development process differ in number of ways. The object-oriented framework is

not completely independent.

**Logical versus physical system structure**

BETA is a language describing system (program execution) consisting objects and

patters, represent phenomena and concepts from application domain, others for

implementation purposes. BETA objects and patterns provide logical structure of a

system; BETA language provide mechanisms describing logical structure.

 BETA program (or BETA description) constructed in form of text in one or more files.

Program can exist number of variants for different computers, and exist in various

versions. Many consist of modules from library; may use many different programs.

BETA language don’t have mechanisms describing physical organization of BETA

program in terms of files, variants, and versions, etc., the physical structure of program

text considered independent of logical structure of program. Languages provide

language elements handling physical structure, e.g. modules are divided into interface

and implementation modules. This mechanisms are not part of BETA. It’s a language-

independent technique for organizing physical structure of a program text been

developed. Technique based context free grammar of language, correct sequence of

terminal and nonterminal symbols (called **sentential form**) of grammar be a module.

**1 of 7**

This technique used language other than BETA.

 The aspects of physical structure other than organization of program text. BETA

system have number of objects exacting actions, the actions expected objects may take

place concurrently.

 Some objects of BETA system *transient*  the sense that only exist while program is

executing. Object are persistent in sense they store on secondary storage like disk and

survive program that created them. Can read and used by other programs. Separation

of objects into transient and persistent object not part of BETA language, handled by

Mjolner BETA system. Object-oriented database system supporting client and servers

currently being developed top of persistent object store the Mjolner BETA System.

 Modern computer hardware consists large number of processors connected to

communication media, example set of workstations connected to local area network.

The BETA program may realized through time sharing on single processor, distributing

BETA objects on number of processors. BETA constructs the logical structure of

concurrent object and physical structure mapping concurrent objects onto several

processors considered independent of language. The Mjolner BETA System contain

support for distribution computing in BETA.

 The mapping of BETA system, into a process generator (computer hardware)

splitting objects in transient and persistent objects and distribution objects to several

processors

 During design and implementation, programmer should be explicit about physical

organization of program text and concerned with organization of persistent and

distributed objects.

**BETA**

**1 of 8**

BETA a modern language in Simula tradition. That supports object-oriented on

programming and contains facilities for procedural and function programming.

 BETA replace classes, procedures, and functions types by single abstraction

mechanism called **pattern**. Virtual procedures to virtual patters, streamlines linguistic

notions like nesting and block structure, provide framework for sequential, co-routine

and concurrent execution. The language smaller than Simula in spite being more

expressive.

 Mjolner BETA System a software environment development supporting BETA

language incudes a implementation of BETA language. System includes number of

other tools, a collection of large libraries and frameworks greatly enhance BETA’s

usability providing large number of predefined patterns and objects.

 Mjolner BETA System originally part of Nordic Mjolner project, produced large

number other results. (Knudsen *et al.,* 1992) collection of project’s results.

**Introduction to Basic Concepts**

 A computer executing program process consisting of various phenomena (example

phenomenon an object representing bank account of customer). Object represents

certain properties the real bank account, like its balance, deposits and withdrawals

preformed on the account. Objects reside in computer’s memory.

 Real world, customers and bank clerks perform *actions* change *state* of various bank

accounts. Certain point, balance of given account be DKK 500, a deposit of DKK 105

change its balance DKK 5105. Deposit example action performed in a bank; example of

other withdrawal of money, computation of interest, opening and closing accounts, etc.

Important to aware fact that actions ordered in time – most banks you deposit some

money before can carry out withdrawal.

 Process generated by computer phenomena representing action and states. State of

**1 of 9**

bank account represented object represented bank account accounts changing the

state bank account are represented by actions executed by the computer.

 Process generated by computer executing program called *program execution*.

Program executions is a class of processes called *information processes*. Production

process car in a factory, processing of customer orders in a company, money flow

between banks viewed as information processes.

 (Lehrmann & Moller, 1993)

**Procedural programming**

When computers invented they viewed as programmable calculator some people still

have this view.

 Simple calculator, may consist of register storing a value, number of operations like

add, sub, mult and div. enter a number into register and modify value using operations.

List of operations for a simple calculator:

 Enter V

 Add V

 Sub V

 Mult V

 result

Operations enter, add, sub, mult, div and result correspond to buttons on calculator; V

corresponds to number entered into a pad. The physical layout of calculator not concern

about, only interested in the functionality.

 Instead of one register calculator have several registers, intermediate results of

calculation be stored in registers -- only one register user would write down

intermediate results on paper. Assume calculators is extended with registers R0, R1,

R2, …Rn, previous operation operate R0. Following new operations:

 Enter V Ri

**1 of 10**

 Add Ri Rj

 Sub Ri Rj

 Mult Ri Rj

 Div Ri Rj

 Copy Ri Rj

Operands Ri and Rj correspond buttons selecting a register.

 Calculator with fixed number registers and operations has limited scope applications.

User carry out same procedure over and over again on different data, perhaps carrying

out same sequence of operations. Lead to a *programmable calculator*. Data registers

and operations, programmable calculator has store sequence of operations may be

stored:

 define Pi Op1; op2; …end

 call Pi

The *define* and *call* operations correspond bottoms for calling procedures. Pi be number

or another unique identification procedure being defined. Programming calculator be

used in following way:

 define P1: copy R0 R1; mult R0 2; add R0 R1 end

 enter 100

 call P1

 return

Store used storing procedures also used storing values, i.e. move value from a register

to store, and vice versa. It possible save large number of intermediate results.

Operations extended with control flow operations, possible select between sub-

sequences of operations repeat execution of sub-sequence. Example flow operations:

 L:

 goto L

**1 of 11**

 If Ri=0 goto L

**Computers** (Shouhong, 2014)

A computer general machine that is programmed to carry out computation and data

processing operations. The programs can changed by humans through programming,

computer can solve many problems. The computer has central processing unit (CPU),

interprets and executes programs, and primary memory, store program and data.

Computer system also include secondary memory, input device and out device, see

Figure 1.1. Input device converts human signals and data signals that processed by

CPU. Example input devices keyboard and mouse. Output device converts signals from



CPU into form understandable to human. Example of output devices monitor and

printer. A primary memory, secondary memory also used to store program and data.

The programs and data store in primary memory cannot retained when power is turn off.

Secondary memory retain stored programs and when power is turn off. The CPU

access programs and data faster in primary memory than second memory. Programs

and data in secondary memory read batches into primary memory before used by CPU.

**1 of 12**

**Computer Programming Languages**

Computer programming language artificial language designed to communicate

instructions to computer. Programming languages used create programs that control

computer perform task designed. These are tasks computer carry out:

* Manipulating data and information
* Reading data from and /or writing data to secondary memory or other

 input/out devices

* Presenting data for human through user—computer interface

Many computer programming languages. Each computer programming language has

syntax.

**Software systems**

Software systems in computer structure in layer, Figure 1.2 below application software

build by software developer using high-level programming language easily understand

and use.



 (Shouhong, 2014)

**1 of 13**

Programs in high-level programming language cannot executed by computer

unless programs translated into machine executed code (strings of binary digits).

Translate program in high-level programming language into machine executable code,

called compiler or interpreter for high-level language, must applied, show Figure 1.3 a

program in high-level programming language translated the machine-executable code,

can used infinite number of times.

**Programmer** **Computer**

 Edith program

 Using program editor Execution

Machine Code

Translation

Programs in high-level computer programming language

**Figure 1.3 Translation of computer programs**

Program in high-level programming language has syntax error, translation will fail and

machine-executable code not be generated. A program without syntax error have

logical error, or semantic error, and final execution result be incorrect. To be sure

program executed correctly, computer programmer do the following three tasks.

1. *Procedural language versus makeup language*. A procedural language capable

commanding a computer to carry out arithmetic or logical operations. Programming

language except HTML and XML are procedural languages. Makeup language used

annotating document (or data set) in syntactically distinguishable text. HTML and XML

markup languages.

**1 of 14**

2. *Function-oriented language versus object-oriented language*. Function-oriented

language function as modules. C is typical function-oriented language. A computer l

language can be blended language of function-oriented and object-oriented language,

like JavaScript and VB.NET.

3. *Client-side language versus server-side language*. Client-side language used to

create computer programs that executed the client side on web. JavaScript and HTML

typical client-side languages. Programs in server-side language like PHP and AS.NET

executed by the Web server have greater access to information and functional

resources available on server response to client’s request.

 (Shouhong, 2014)

**Generation of Programming Languages** (Beynon, 1998)

The consensus there at least three generations of programming languages:

1. ***Machine code***. That is earliest form of programming language, only one step

 removed from binary code by machine to perform instructions.

2. ***Assembly language***. The first abstract detail machine and provide programmer

 with powerful set symbolic instructions to write programs.

3. ***High-level languages*** ( known as thirund generation languages). These meant

 general-purpose programming languages removed from machine implementation.

 Language can divided into groupings such as imperative languages (FORTRAN,

 COBOL, C), functional languages (LIST), logic programming languages

 (PROLOG), and object-oriented languages (Smalltalk, C++). Imperative languages

 are widely used for information systems development, object-oriented languages

 are beginning to have influence in the development domain. Figure 1.4 show family

 tree of high-level languages.

**1 of 15**



**Figure 1.4 Family Tree of High-Level Languages**

**The Major Constructs of Structured Programming Languages**

The building blocks most structured programming languages. A simple dialect of

language Pascal, language used in education circles and derivative of ALGOL.

 Most programming languages build out two constructs:

**Statements**

Statements the basic instructions of high-level languages. Statements command line

build from mixture of keywords, variables and constants. Examples valid Pascal

statements are:

balance :=balance + credit;

READ (credit);

WRITELN (Balance is’ , balance);

First statement assigns summation value in two placeholders or variable, balance and

credit, placeholder on left the ‘:=’ sign. Second statement read value file into variable.

third statement writes string to terminal screen.

**1 of 16**

**Control Structures**

Control structures, control the flow execution statements. Control structures come in

three forms: sequences; conditions; and loops.

**Sequences:** logical set statements. Pascal is encase sequences in keywords BEGIN

and END, called block, e.g.

BEGIN

 Balance :=balance + credit;

 WRITENLN (‘Balance is’, balance

END

BEGIN

 Balance := balance – debit;

 WRITELN (‘Balance is’, balance);

END;

**Conditions:** allow selection alternatives. There three forms of conditions:

1. Single-branched

IF credit>0 THEN balance :=balance + credit;

2**.** Double-branched

IF transactionType = ‘C’ THEN

 Balance :=balance + credit

ELSE

 Balance :=balance – debit;

3. Multiple-branched

CASE transactionType OF

 “C”: balance :=balance + credit;

 “D” balance :=balance – debit;

 “P” WRITELN (‘Balance is’, balance);

END;

**Countable loop**.

FOR count := 1 to 10 DO

 BEGIN

 READ (credit);

**1 of 17**

 Balance :=balance + credit;

 Count :=count + 1

END;

While loop’ Test performed at start each iteration. If specified is true loop continues. If

condition false, loop terminates.

Count := 1

WHILE count<11 DO

 BEGIN

 READ (credit):

 Balance :=balance + credit;

 Count :=count +1

 END;

 (Beynon, 1998)

**1 to 18**

**CONCLUSION**

Object-oriented programming becoming ubiquitous paradigm in modern day computing.

It has many features of structure paradigm, like a structured movement that influenced

approaches to analysis, design and programming, the object-oriented has been an

impact upon systems analysis, system design, programming and recently data based

systems.

**1 of 19**

**BLIOGRAPHY**

Beynon, D. P. (1998). *Information System Development.* London: MACMILLAN PRESS Ltd.

Lehrmann, M. o., & Moller, P. B. (1993). *Object-Oriented Programming in the Beta Programming Language.* USA: Ass Computer Machinery.

Shouhong, W. H. (2014). *Programming Language MIS: Concept and Practic .* USA: Taylor & Francis Group CRC Press.

along the front end or back end. (Jon, 2007)

**1 of 17**

**CONCLUSION**

**1 of 18**

**BLIOGRAPHY**

John, H., & David, P. (2019). *Computer Architecture.* USA: Morgan kaufmann.

Jon, S. (2007). *Introduction to Microprocessors and Computer Architecture.* USA: No Starch Press, Inc.

Richard, D. (2015). *80x86 Assemble Language and Computer Architecture.* USA: Jones & Bartlett Learning.