

Unit_2

(Object Oriented Calculus)

Overall Course Description

This course (Unit_1-Unit_10) provides the basics of object oriented programming descendants based on molecular biology. Also, it offers basic skills in problem solving and object-oriented programming using any high-level language an example would be Java. Topics include algorithm development, simple data types, expressions and statements, program flow control structures, objects, methods, and arrays.

Overall Learning Goals

At the completion of this course, the students will be able to understand the origin of object oriented programming, apply, design and development principles in the construction of software systems of varying complexity, read a problem description, design an algorithm to solve the problem and document their solution. Also, design, implement, and document applications using Object-Oriented Programming concepts.

Unit_2 Objective

- Understand the concept of objects
- Understand the concept of methods
- Understand the concept of data types
- Understand the concept of decisions
- Understand the concept of iterations

The goal of Unit_2 is to learn and understand how Object Oriented Biology, OOB transitions into Object Oriented Calculus, OOC. Furthermore, applying Object Oriented Calculus to the basic foundations of any object oriented language.

In addition, below are all the competencies listed that one should master after studying Unit_2.

Competencies for Unit_2

A. PLAN:

- A.1. Objects
- A.2. Methods
- A.3. Data Types
- A.4. Decisions
- A.5. Iterations

Competence Scope

Identifier: A.1

Name: Objects

Description: An ability to design, implement and evaluate a component like an object to meet desired needs in Object Oriented Calculus.

Skills Examples:

S1: using objects for understanding objects in Object Oriented Calculus.

Knowledge Examples:

K1: objects in Object Oriented Calculus.

Proficiency Levels:

P1: using objects for understanding objects in Object Oriented Calculus with an application to any Object Oriented Language.

Competence Scope

Identifier: A.2

Name: Methods

Description: An ability to design, implement and evaluate a component like a method to meet desired needs in Object Oriented Calculus.

Skills Examples:

S1: using methods for understanding methods in Object Oriented Calculus.

Knowledge Examples:

K1: methods in Object Oriented Calculus.

Proficiency Levels:

P1: using methods for understanding methods in Object Oriented Calculus with an application to any Object Oriented Language

Competence Scope**Identifier:** A.3**Name:** Data Types**Description:** An ability to design, implement and evaluate a component like a data type to meet desired needs in Object Oriented Calculus.**Skills Examples:**

S1: using methods for understanding data types in Object Oriented Calculus.

Knowledge Examples:

K1: data types in Object Oriented Calculus.

Proficiency Levels:

P1: using methods for understanding data types in Object Oriented Calculus with an application to any Object Oriented Language

Competence Scope**Identifier:** A.4**Name:** Decisions**Description:** An ability to design, implement and evaluate a process like a decision to meet desired needs in Object Oriented Calculus.**Skills Examples:**

S1: using decisions for understanding decisions in Object Oriented Calculus.

Knowledge Examples:

K1: decisions in Object Oriented Calculus.

Proficiency Levels:

P1: using decisions for understanding decisions in Object Oriented Calculus with an application to any Object Oriented Language

Competence Scope**Identifier:** A.5**Name:** Iterations**Description:** An ability to design, implement and evaluate a process like an iteration to meet desired needs in Object Oriented Calculus.**Skills Examples:**

S1: using iterations for understanding iterations in Object Oriented Calculus.

Knowledge Examples:

K1: iterations in Object Oriented Calculus.

Proficiency Levels:

P1: using iterations for understanding iterations in OOC with an application to any Object Oriented Language

Unit_2 Contents

- 2.0 Object Oriented Calculus
- 2.1 Objects
- 2.2 Self-Check Questions for the Objects Sub-Section
- 2.2 Methods
- 2.3 Self-Check Questions for the Methods Sub-Section
- 2.4 Data Types
- 2.5 Self-Check Questions for the Data Types Sub-Section
- 2.6 Decisions
- 2.7 Self-Check Questions for the Decisions Sub-Section
- 2.8 Iterations
- 2.9 Self-Check Questions for the Iterations Sub-Section
- 2.10 Answers to Self-Check Objects Sub-Section
- 2.11 Answers to Self-Check Methods Sub-Section
- 2.12 Answers to Self-Check Data Types Sub-Section
- 2.13 Answers to Self-Check Decisions Sub-Section
- 2.14 Answers to Self-Check Iterations Sub-Section

Table of Contents

2.0 Object Oriented Calculus, OOC	1
2.1 Objects	2
2.2 Self-Check Questions for the Objects Sub-Section.....	3
2.3 Methods.....	4
2.4 Self-Check Questions for the Methods Sub-Section	6
2.5 Data Types	7
2.6 Self-Check Questions for the Data Type Sub-Section	8
2.7 Decisions	9
2.8 Self-Check Questions for the Decision Sub-Section	10
2.9 Iterations	11
2.10 Self-Check Questions for the Iterations Sub-Section	14
2.11 Answers to Self-Check Objects Sub-Section	15
2.12 Answers to Self-Check Methods Sub-Section.....	16
2.13 Answers to Self-Check Data Types Sub-Section	17
2.14 Answers to Self-Check Decisions Sub-Section.....	18
2.15 Answers to Self-Check Iterations Sub-Section	19

2.0 Object Oriented Calculus, OOC

In this sub-section OOB is transformed in OOC meaning a concrete transition is demonstrated from a cell to an object. The relationship between cell and the receptor is explained using an object and a method. Furthermore, the update process is clarified meaning how an object contents can be modified via a method. Finally, the if-statement and various loops are developed based on OOB.

2.1 Objects

When humans experience new events, they can be classified as new memories or objects (cells in OOB). The objects are stored in our mind and are also known as the mental space. The mental space can be considered as HMS, which consists partially out of the WM and the LTM and is responsible for producing the codec process meaning it encodes new objects from the WM into the LTM and decodes stored objects from the LTM into the WM. The formation of new memories can be compared to creation of new objects in an OOL, such as Java where the new operator creates the object in the computer physical memory also known as a block of memory or storage space.

Objects resemble cells, which are created from a class that acts as a blueprint. Also, objects have attributes and behavior that make up the unique internal state, which are defined by classes. Just like the cells have internal processes (inside the nucleus), which interact within the boundaries of the plasma membrane or cell wall, objects have private variables which are initialized inside the constructor methods (nucleus) and cannot be modified directly. In both cases it is known as encapsulation or data security.

An object is created from a class cell using the construction **new cell**. The new cell assigns an attribute record and returns a reference or position to it. The attribute record holds the initial values and the methods code specified by the cell. Once the new cell is executed the cell object is generated and a reference to the attribute record is created. It can be classified as an object of class cell or an instance of class cell.

The creation of the object cell can be achieved by using the following notation:

```
var cellaCell: InstanceTypeOf(cell) := new cell;
```

The expression can be divided into three parts such that the first part contains the variable var called cellaCell or object name. The second part has the **InstanceTypeOf(cell)** is the type of object of class cell and the third part entails the new operator and the cell class. Part two is associated to part three by using the := meaning equal by definition and part two is connected to part one by : meaning true to that.

Another way of writing the same expression can be done in the following way:

```
var cellaCell:cell := new cell;
```

The cell still has the same meaning as the **InstanceTypeOf(cell)** and that is the type of the object created from the cell.

2.2 Self-Check Questions for the Objects Sub-Section

1. New events can be classified as what?
2. What parts does HMS consist of?
3. What does a codec process do?
4. What can formation of new memories be compared to?
5. What does the new operator do?
6. What is the unique internal state of an object made of?
7. Cells have internal processes, which interact within the boundaries of what?
8. Where are the object private variables initialized?
9. What are the three parts that an object is made of?
10. Objects resemble what?
11. What is an object?

2.3 Methods

For the cell or object, which has behavior to be able to communicate with other cells, it will make use of cell receptors or functions/methods as they are known in OO programming languages. The method carries out an action, which is able to make changes to the cell organelles (instance fields) or object contents therefore there exists a relationship between a cell and a receptor meaning modifying the contents of the cell is done through the receptor and can be expressed as **cell.receptor** or in OO programming as **object.method**.

When OO languages were created the attempt was to mimic nature processes or OOB and needed to be converted into a scientific form like OOC for being able to simulate different objects from the real world.

OOC is responsible for explaining how any OOL logic functions. Every OOL has a foundation that makes up the language, which consists out of objects (cells), functions (receptors), and classes (blueprint). As stated, before an object is a block of memory that allows us to simulate real world objects using any OO programming language. Furthermore, an object has behavior and belongs to a class. A class is a collection of objects and its naming convention should be represented as a noun. The method carries out an action, its naming convention should be represented in a form of a verb and it also belongs to a class. Methods exist in two flavors, such as mutators and accessors. A general diagram is shown below in Fig. 1.

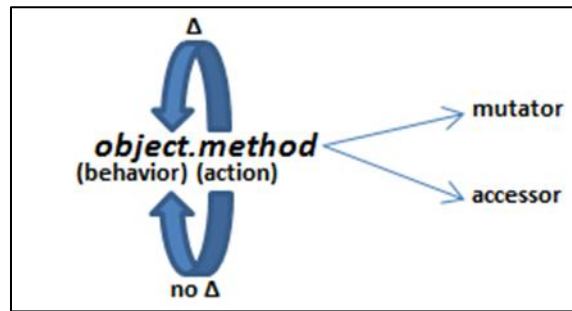


Fig. 1

OOC is a descriptive discipline meaning no problems are solved mathematically instead they are described scientifically using Sigma ζ -calculus or calculus of objects. The OOP concepts and logic descends from OOB and is transformed into OOC, which is a procedural language. In OOC the formula **object.method** is represented as **o.l** where **o** is the object and **l** is the method. The explanation of the formula **o.l** will demonstrate where all the OO languages philosophies on objects and methods associations come from. The true meaning of this formula can be stated as an invocation of method **l** of object **o** meaning that when **l** is applied on the **o** a modification of the **o** contents occurs. By looking at this process in more details the real meaning of **o.l** can be explained with the following expression **o.l $\Leftarrow \zeta(x)b$** . Functions or methods are recognized if they have the basic calculus structure **f(x)** where **f** is the name of the function and **x** is the input or parameter. The same analogy can be applied to the expression **$\zeta(x)b$** . The Greek letter Sigma ζ represents a function name with an input **x** and **b** represents the body of the function. Putting it together the expression **$\zeta(x)b$** can be read as method ζ with a parameter **x** and body **b**. The leftwards double arrow \Leftarrow means update. The full expression **o.l $\Leftarrow \zeta(x)b$** can be read as update of method **l** of object **o** with method **$\zeta(x)b$** meaning the update produces a copy of **o** and the method **l** is replaced with **$\zeta(x)b$** .

It is possible to transform the OOC expression **o.l $\Leftarrow \zeta(x)b$** into an OOP language. In order to show the true purpose of the OOC expression, it will be necessary to provide a detailed explanation of objects and methods and their relationship to each other and to OOC.

Furthermore, it is possible to explain via OOC the creation of data types, decisions, and iteration in OO programming languages.

2.4 Self-Check Questions for the Methods Sub-Section

1. What does the method carry out?
2. Name the three most important foundation that any OOP language has?
3. What is a class?
4. What are the two methods that exist in OO languages?
5. What is the naming convention that should be used for a method?
6. What is the method convention that should be used for a class?
7. What kind of a discipline is OOC?
8. What does the leftwards double arrow mean?
9. By what means do objects get updated?
10. What kind of science discipline is OOC based on?

2.5 Data Types

The DNA (**responsible for the cell functionality inside the nucleus**) is a genetic program of a cell, which leads to a cytoplasm (**fluid in the cell, mostly water**). It is like a data type that can be compared to a computer instruction, which can modify data therefore at least one data type like integer must be available. The data type can be represented by metabolites (**intermediate products**) and modified by biochemical reactions.

The data type (**metabolite**) like an integer can be represented by OOC in the following way:

var number: Integer := 0;

The **var** stands for variable and number is the name of the variable. The Integer is the type and the 0 is the contents of the type, which is the same in Java. The : means true to that, := means equal by definition, and ; means end of the expression, therefore we can state that the contents is stored in the variable, which is a storage location in the computer's memory. The expression above can be classified as a data type.

2.6 Self-Check Questions for the Data Type Sub-Section

1. What does an initialization of a data type mean?
2. What is a metabolite?
3. What is a data type?

2.7 Decisions

As stated previously **if** is a keyword, and is used to implement a decision, which comes from OOB and can be expressed with the following OOC as shown below in Fig. 2.

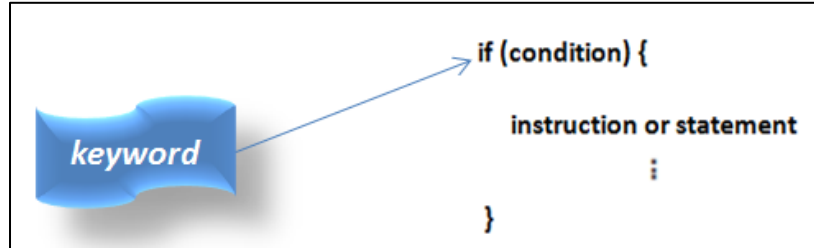


Fig. 2

From OOB the operon can be represented as an if statement meaning instruction S will be executed if condition B is true, otherwise S will not be executed (**if B then S**). By focusing on the E. coli bacterium, which regulates its own synthesis. The boolean value of condition B can be classified by the state of the operator, which is true if **Operator_X** gene is free or false if **Operator_X** gene is blocked by the repressor. The if statement can be simulated once the operon is blocked after activation of the basic instruction. The synthesis of structure gene S will realize the instruction S and the synthesis of **Regulator_X** gene will block the synthesis of operon L14.

The **Operator_X** gene is the condition and can be either true or false and the instruction or statement is the structure gene S and so the if statement can be classified as the following:

If the condition is true the statements will be executed inside a block {...}, but if the condition is false the statements won't be executed meaning **if B then S**.

2.8 Self-Check Questions for the Decision Sub-Section

1. What are the two Operator_X conditions?
2. In programming what is a decision known as?
3. In order to simulate a decision, the operon must in what mode?
4. What blocks the Operator_X?

2.9 Iterations

By deleting the **Regulator_X** gene, which is found inside the operon L14 the while statement can be imitated meaning let S be an instruction and B a condition, which can be true or false, therefore the instruction S will be executed as long as B is true (**while B do S**).

The trp is an α -amino acid that is used in the biosynthesis of proteins where trp operon is a group of genes that is transcribed together—that codes for the components for production of trp and therefore structure genes S and the boolean value of condition B can be true that is operator is free or false operator is blocked. Suppose the operon represents the state true the basic instruction is activated, and it will be active until the operator gene will be blocked and this logic simulates the while statement shown below in Fig. 3.

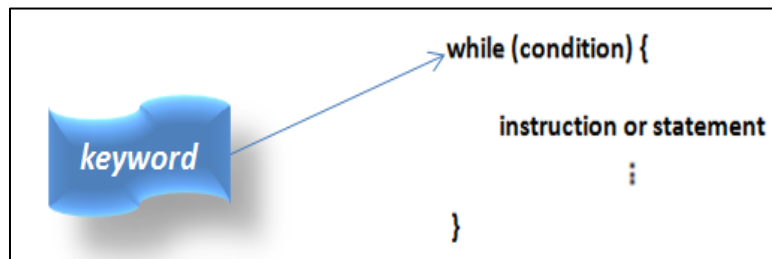


Fig. 3

From OOB the **Operator_X** is the condition and can be either true or false and the instruction or statement is the structure gene S therefore as long as the condition is true the statement will be executed inside the block {...} meaning **while B do S**.

Furthermore, it is possible to enhance the while loop into a do-while loop when execution of a loop must occur at least once shown below in Fig. 4.

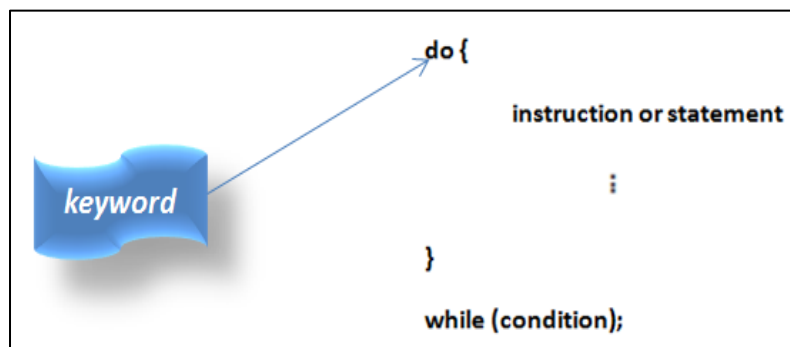


Fig. 4

The statement or instruction is executed first. Furthermore the condition is tested once the statement or instruction is executed meaning the statement or instruction is executed at least once meaning the operator is the B condition and can be either true or false and the instruction or statement S is the structure gene S therefore as long as the condition is true the instruction or statement will be executed inside the block {...} again.

The while loop can be extended into a for loop, it's a special form of the while loop by adding two minor things to it. First, initializing or defining a variable such as **i = start** where **i** is the variable, **start** is the contents, and **=** is the assignment

operator meaning start is stored in **i**. Second, the increment value will be added to the code below in the following way **i = i+1** meaning add one to the variable **i** and store the result in **i**. The next time the stored variable **i** will add one again to the stored variable **i** and so on. Also, it is possible to decrement by one like so **i = i-1**. In general, it is possible to increment and decrement by any number. There is shorthand for the expressions, which can be used such as increment by one **i++** and decrement by one **i--**.

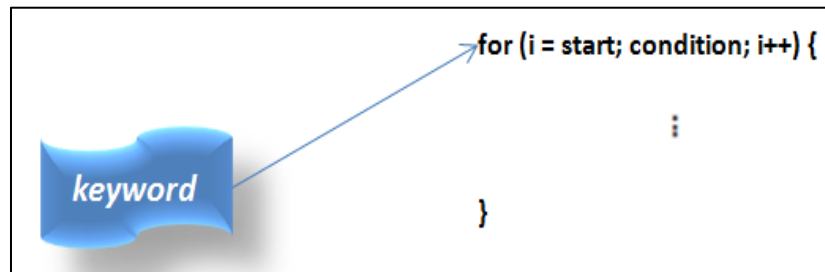
```
i = start;

while (condition) {

    i++;

}
```

The while loop has a starting point **i = start**. While the condition is true an increment by one in the while loop body {...} will occur. The extended while loop can be transformed into a for loop shown below in Fig. 5.



```
for (i = start; condition; i++) {

    :

}
```

Fig. 5

The for loop has a starting point **i = start**, a **condition**, and an **increment**. Furthermore, the variable **i** is applied to the condition by comparison and if it is true the for-loop body {...} will be executed and afterwards incremented, but if it is false the for-loop body will not be executed, and no increment will occur.

In this sub-section the relationship of a cell and receptor was related to object and method. Also, the update process was discussed that is how a method updates the object. In addition, data types, if statements and various loops were presented. In the next sub-section OOC will be applied to an OOP language like Java.

2.10 Self-Check Questions for the Iterations Sub-Section

1. Where is the Regulator_X found? Inside the operon L14
2. What is the minimum times the do-while loop will execute? Once
3. What loop is the basis for the for loop? The while loop

2.11 Answers to Self-Check Objects Sub-Section

1. New events can be classified as what?
New memories or objects
2. What parts does HMS consist of?
Working Memory, WM and Long Term Memory, LTM
3. What does a codec process do?
It encodes new objects
4. What can formation of new memories be compared to?
Creation of New objects
5. What does the new operator do?
Creates an object in the computer's memory
6. What is the unique internal state of an object made of?
Attributes and behavior
7. Cells have internal processes, which interact within the boundaries of what?
Plasma membrane (cell wall)
8. Where are the object private variables initialized?
In the constructor
9. What are the three parts that an object is made of?
Object name, type of the object, object itself
10. What do Objects resemble?
Cells
11. What is an object?
Block of memory

2.12 Answers to Self-Check Methods Sub-Section

1. What does the method carry out?
Action
2. Name the three most important foundation that any OOP language has?
Objects, Methods, Classes
3. What is a class?
Collection of Objects
4. What are the two methods that exist in OO languages?
Accessors and Mutators
5. What is the naming convention that should be used for a method?
Verb
6. What is the method convention that should be used for a class?
Noun
7. What kind of a discipline is OOC?
Descriptive Discipline
8. What does the leftwards double arrow mean?
Update
9. By what means do objects get updated?
Via methods
10. What kind of science discipline is OOC based on?
Molecular Biology

2.13 Answers to Self-Check Data Types Sub-Section

1. What does an initialization of a data type mean?

It means a data type, name, and contents

2. What is a metabolite?

It is a data Type

3. What is a data type?

It tells you what one can do with it in a computer program

2.14 Answers to Self-Check Decisions Sub-Section

1. What are the two Operator_X conditions?
True and False
2. In programming what is a decision known as?
If statement
3. In order to simulate a decision, the operon must in what mode?
In the false mode
4. What blocks the Operator_X?
The repressor

2.15 Answers to Self-Check Iterations Sub-Section

1. Where is the Regulator_X found?
Inside the operon L14
2. What is the minimum times the do-while loop will execute?
Once
3. What loop is the basis for the for loop?
The while loop