Unit_3
(Object Oriented Calculus Application)

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 a high-level language such as 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 understands 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_3 Objective

-Understand the concept of construction of objects
-Understand the concept of construction of methods
-Understand the concept of integration of object oriented programming concepts

The goal of Unit_3 is to learn and understand how construction of objects, methods, and integration of objects oriented programming concepts occur.

In addition, below are all the competencies listed that one should obtain after studying Unit_3.

Competencies for Unit_3

A. PLAN:

- **A.1.** Construction of Objects
- **A.2.** Construction of Methods
- **A.3.** Integration of Object Oriented Programming Concepts

Competence Scope
Identifier: A.1
Name: Construction of Objects
Description: An ability to design, implement and evaluate a component like an object to meet desired needs in Object Oriented Calculus with an application to OOP languages like Java.

Skills Examples:

S1: using construction of objects for understanding construction of objects in OOP

Knowledge Examples:

K1: construction of objects in OOP languages like Java

Proficiency Levels:

P1: using construction of objects for understanding construction of objects in OOP language like Java

Competence Scope
Identifier: A.2
Name: Construction of Methods
Description: An ability to design, implement and evaluate a component like a method to meet desired needs in Object Oriented Calculus with an application to OOP languages like Java.

Skills Examples:

S1: using construction of methods for understanding construction of methods in OOP

Knowledge Examples:

K1: construction of methods in OOP languages like Java

Proficiency Levels:

P1: using construction of methods for understanding construction of methods in OOP language like Java

Competence Scope
Identifier: A.3
Name: Integration of Object Oriented Programming Concepts
Description: An ability to integrate object oriented programming concepts and their relationship to OOP languages like Java.

Skills Examples:

S1: using integration of object oriented programming concepts for understanding integration of object oriented programming concepts in OOP

Knowledge Examples:

K1: integration of object oriented programming concepts in OOP languages like Java

Proficiency Levels:

P1: using integration of object oriented programming concepts for understanding integration of object oriented programming concepts in OOP language like Java

Unit_3 Contents

Object Oriented Calculus Application

In this sub-section OOC will be transformed into application to programming where major logical parts that form any OOP language will be explained using Java syntax.

Construction of Objects

Previously stated objects resemble cells and are created from a class or a blueprint. Also, they have attributes and behavior known as internal state defined by classes. Cells have internal processes which interact within the limitations of the plasma membrane and objects have private variables which are initialized inside the constructor methods and cannot be modified directly known as encapsulation.

The simplest way would be start out with the object and from previously it is known that an object is a block of memory, space or an entity, which can be expressed for example in Java code. The creation of a bank-account object using Java can be accomplished with the following syntax **BankAccount anAccount = new BankAccount();**. This expression can be divided into a left-hand side and a right-hand side expression. The right-hand side expression contains the **new** operator that creates the object and the **BankAccount()** with parentheses meaning it's a class and a call to the constructor or **new cell**. A constructor allows for the initialization of the object and contains instance fields, which are protected variables that hold the object contents like **currentBalance**. The constructor has the same name as the class in this case it is **BankAccount** and it has an explicit parameter named **aBalance** which is a data type **double**. Notice that **aBalance** contents is stored in the instance field **currentBalance**.

```
public BankAccount ( double aBalance){

        currentBalance = aBalance;

}
```

Classes are capitalized in Java. The left-hand side contains the **BankAccount** or **IntanceTypeOf(cell)**, which is a class and the name of the object the **anAccount** or **var cellaCell**, which can be classified as a reference. The assignment operator **=** means that the memory location of the object is stored in the reference. The expression **BankAccount anAccount = new BankAccount();** can be compared to the object **o** in the OOC. A visual object summary is shown below in Fig. 6.

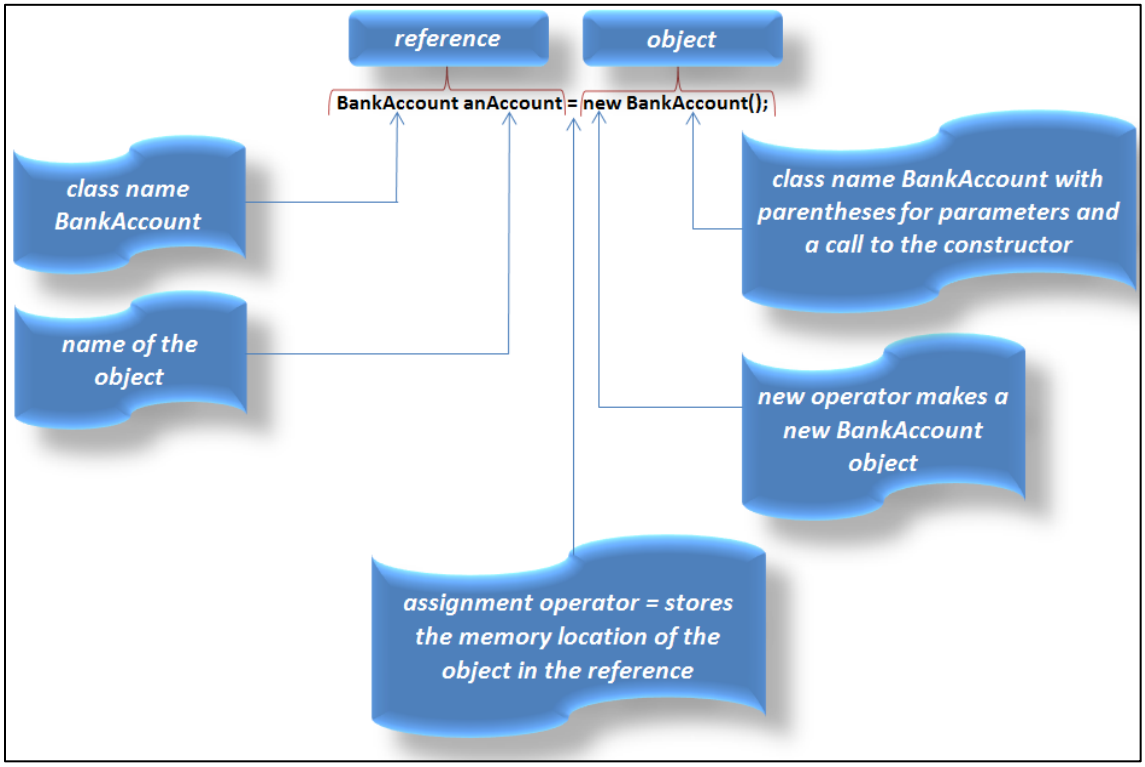

Fig. 6

Any object can be created using this structure with Java code. Again, **BankAccount anAccount = new BankAccount();** is the equivalent object **o** in OOC shown in the table below. Furthermore, this object has behavior and belongs to the class **BankAccount**.

Java Code, OOL	Procedural Language, OOC
BankAccount anAccount = new BankAccount();	**object o**

Self-Check Questions for The Construction of Objects Sub-Section

1. What is an object?
2. What does the new operator do?
3. What is a reference?
4. What is a constructor?
5. Where does the object store its contents?
6. What is an instance field?
7. What does an object have?

Construction of Methods

The cell or object has behavior and is able to communicate with other cells via cell receptors or functions/methods as known in OO programming languages. The method carries out an action, which makes 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**.

In order to make a change to the object a method is required, that is method **l** from OOC. It has an action and can be identified using a calculus structure **f(x)**. This calculus structure can be transformed into Java code as maybe a deposit method in the following way **deposit()**. Looking closely at the method it contains the same structure as the calculus function **f(x)**. A visual method summary is shown below in Fig. 7.

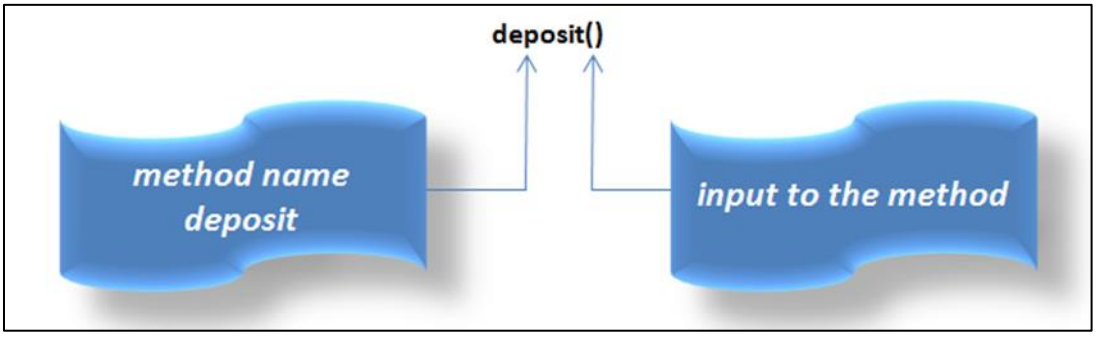

Fig. 7

Any method can be created with this structure using Java code. Again, deposit() is the equivalent method l in OOC shown in the table below.

Java Code, OOL	Procedural Language, OOC
deposit()	**method l**

There is an association between the object and the method that is **object.method** or **o.l**. This association can be transitioned into Java code as **anAccount.deposit()** where **anAccount** is the name of the object and **deposit()** is the name of the method. A visual object dot method summary is shown in the table below.

Java Code, OOL	Procedural Language, OOC
anAccount.deposit()	**o.l**

The expression **ς(x)b** can be transformed into Java code in the following way:

```
public void deposit (double anAmount) {

        double tempBalance = anAmount + currentBalance;
        currentBalance = tempBalance;
}
```

The **deposit(double anAmount)** means the following: **deposit** is the method and is equivalent to the **ς** in OOC and **double anAmount** is the explicit parameter or the input to the method meaning **double** is the data type and the **anAmount** is the name of the data type equivalent to the **x** in OOC. The body of the method is the code between the parentheses {…} also known as a block and is equivalent to the body **b** in OOC.

A visual method with input summary is shown in the table below.

Java Code, OOL	Procedural Language, OOC
deposit(double anAmount)	**ç(x)**
double tempBalance = anAmount + CurrentBalance; **currentBalance = tempBalance;**	**body *b***

The method **deposit()** is a mutator method, because it contains the keyword **void**, which specifically states that it does not return a value and it's a **public** method, since it has an access specifier **public** meaning it can be used by other classes.

Another type of a method is an accessor method meaning it does not change the contents of the object but instead displays the contents of the object.

```
public double getCurrentBalance(){

        return currentBalance;

}
```

An accessor method always has a return type in this case it is a double. Furthermore, the method is created by using the get command with a name of a method that is **getCurrentBalance()**. The **return** statement instructs the method to terminate and return the output of the method meaning the contents of the object, because following the return statement is the instance field **currentBalance** and that is where the object stores its contents.

Self-Check Questions for The Construction of Methods Sub-Section

1. What does a method carry out?
2. What is a connection operator that is used between and object and a method?
3. What is an explicit parameter?

Integration of Object Oriented Programming Concepts

Let's demonstrate the functionality of the expression **o.l⇐ ς(x)b** via Java syntax that is shown below.

```
public class BankAccount {

        public BankAccount ( double aBalance){

                currentBalance = aBalance;

        }

         ⋮

        public void deposit (double anAmount) {

                double tempBalance = anAmount + currentBalance;
                currentBalance = tempBalance;

        }

        public double getCurrentBalance(){

                return currentBalance;

}

        private double currentBalance;

}

public class BankAccountTester {

        public static void main(String args[]) {

                ⋮

        BankAccount anAccount = new BankAccount();
        anAccount.deposit(500);

        }

}
```

The Java code consists out two classes where the **BankAccount** class is a sub-class or a module and holds the code of the program and the main class **BankAccountTester** is the main class, which contains the method **main** that is responsible for testing the program for correctness and executing it. The sub-class connects to the main class by using the object such as **BankAccount anAccount = new BankAccount();** which matches the name of the sub-class **BankAccount** and its contents is stored in the instance field in the sub-class **private double balance** where **private** is the access specifier and allows modification to the variable balance through the method **deposit**. Its data type is a **double** and name is **currentBalance** can be classified as a declaration, since data type and name means exactly that. The line **anAccount.deposit(500)** is in the main class the equivalent to the OOC **o.l**.

The **public void deposit(double anAmount){ …}** and the code inside the parentheses is the equivalent of the expression **ς(x)b** method with parameter **x** and body **b**. The expression **o.l⇐ ς(x)b** operates on two classes, the sub-class **BankAccount** where **ς(x)b** is found and the main-class **BankAccountTester** where **o.l** operates and executes the code. Again, the connection between the sub-class and the main class is the object **BankAccount anAccount = new BankAccount();** where the **BankAccount** matches the name of the sub-class.

Self-Check Questions for The Integration of Object Oriented Programming Concepts Sub-Section

1. How does the sub-class connect to the main class?
2. What is the difference between a sub-class and a main class?

This sub-section concludes with an application of OOC to an OOP language like Java, transition from OOC classes, objects, and functions to OOP language like Java.

Answers to Self-Check Objects Sub-Section

1. What is an object?
   Block of memory

2. What does the new operator do?
   It creates the object in the computer's memory

3. What is a reference?
   It is a memory location

4. What is a constructor?
   It allows to initialize the object

5. Where does the object store its contents?
   In the instance fields

6. What is an instance field?
   It is a variable where the object stores its contents

7. What does an object have?
   A behavior

Answers to Self-Check Methods Sub-Section

1. What does a method carry out?
   An action

2. What is a connection operator that is used between and object and a method?
   The dot operator

3. What is an explicit parameter?
   It is an input

Answers to Self-Check Integration of Object Oriented Programming Concepts Sub-Section

1. How does the sub-class connect to the main class?
   It uses the object from the main class

2. What is the difference between a sub-class and a main class?
   The main class compiles and executes the program and the sub-class just contains the code of the program