Encyclopedia of Information Science and Technology, Fourth Edition

Mehdi Khosrow-Pour
Information Resources Management Association, USA

IGI Global
DISSEMINATOR OF KNOWLEDGE

Published in the United States of America by
IGI Global
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Names: Khosrow-Pour, Mehdi, 1951- editor.
Title: Encyclopedia of information science and technology / Mehdi Khosrow-Pour, editor.
Description: Fourth edition. | Hershey, PA : Information Science Reference, [2018] | Includes bibliographical references and index.
Identifiers: LCCN 2017000834| ISBN 9781522522553 (set : hardcover) | ISBN 9781522522560 (ebook)
Subjects: LCSH: Information science--Encyclopedias. | Information technology--Encyclopedias.
Classification: LCC Z1006 .E566 2018 | DDC 020.3--dc23 LC record available at https://lccn.loc.gov/2017000834

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Object-Oriented Programming in Computer Science

Rahime Yilmaz
Istanbul University, Turkey

Anil Sezgin
Yildiz Technical University, Turkey

Sefer Kurnaz
Istanbul Esenyurt University, Turkey

Yunus Ziya Arslan
Istanbul University, Turkey

INTRODUCTION

In computer science, a program is composed of a series of commands, which runs within a computer or an electronic circuit, producing information for users. Programming is a that can help programmers while writing a program. Computer programming is the process of writing an algorithm and, it is also the encoding of the algorithm into a notation that can produce and provide information to the users. It can be classified into two groups, that is, system programming and application programming. System programming is a sub branch of the general programming that is composed of low level instructions which are used to operate and handle computer hardware. Application programming is considered as the improved version of the computer programs which can perform specific tasks for the users. One of the application programming types is the object oriented programming (OOP) which is about how information is represented in human mind.

As a computer programming approach, OOP is useful such that it provides easy modeling in designing and developing real entities. This approach is intended to model the entities and, also, the relationships existing between them. OOP allows programmers to define the required classes to create the objects and to apply modifications (manipulations) on them. It can also supply inheritance, polymorphism and encapsulation features to the developers. With these capabilities, the processed data can be isolated from other redundant applications. Because of its abilities that are readily available to the users, OOP is preferred much more than other available programming languages. The inherent properties of OOP, which do not exist in other application programming, can be stated as modularity, extensibility and reusability. This chapter provides a substantial survey of OOP in computer science.

In this chapter, we have highlighted a number of explanations and reviews that are generally accepted and are in common use in OOP. We explain the heart of this chapter OOP concept in section 1, Object Oriented Programming Features, making up the largest section. Main topic of OOP which are included Inheritance, Polymorphism, Abstraction and Encapsulation titles are explained with details in the subtitles of section 1. In section 2, you can find an example of OOP implementation in Java. There are many kinds of OOP languages in use but in this study, Java was given as a strong example to OOP language (Harel, Marron, & Weiss, 2010). The following section is Future Research Directions which include future works related OOP. The

DOI: 10.4018/978-1-5225-2255-3.ch650

chapter ends with Conclusion section that gives a brief information about this study.

BACKGROUND

Programming paradigm is a fundamental style of computer programming which classifies programming languages. Different programming paradigms were developed by considering the concepts and abstraction which are used to represent the elements of a program, and steps that compose a computation. Some of the programming languages are designed to support one paradigm and some of them support multiple paradigms. Before OOP languages, there are also some other paradigms such as classic programming, modular programming and structural programming (Bista, Bajracharya, & Dongol, 2015). These programming technics were helped the programmers while solving their problems. Depending on improving technology, new structure of OOP has emerged so one of these paradigms is OOP which changed radically the programming paradigm continued until the day it appeared. Software methodology used before OOP referred to by the name of the procedural programming. This methodology was based on advancing codes in a particular direction and calling the common function that is used to reduce the workload. This methodology, used in the software world for a long time, has some difficulties. First of all, application of the procedural programming developed as a whole cannot be divided. So, each developer working on the application has to know almost every building of application. Due to its building as a whole, it is hard to make changes on the application. The reason of these difficulties is that procedural programming is an abstract that is not able to model the real world. The real world can be simulated by programmers thanks to OOP by using objects and classes. Object-oriented approach enables to divide a complex system into smaller parts and manageable modules which makes development process easier to grasp and share among members of a developer team and easier to communicate to users who are needed to provide requirements and confirm how well the system meets the requirements throughout the process (Dennis, Wixom, & Tegarden, 2015). Thus, OOP enables modularity and abstraction with increased code understanding, maintenance and expansion. OOP is formed by the collection of objects which communicate with each other in order to perform tasks. This communication is based on messages in OOP, and objects are created from classes.

Class is a blueprint which is used to define objects describing the contents of the objects itself. It is a user-defined prototype for an object that defines a set of attributes and methods which characterize any object of the class. These attributes are data members or variables (static attributes), and methods are dynamic behaviors, also called member functions. Data members and member functions are called class members. Attributes, which are attached to the classes, store information about the object (Robson, 1981). Data member has a name and a type; it holds a value of that type. Member function receives parameters from the caller (if it's required), performs the tasks which are defined in the function body and returns result or void to the caller.

In most of the programming languages, class keyword is used to define a class. Class declaration must contain the name of the class which programmer declares. Basic class declaration looks like this:

```
Class NameofClass {
...
}
```

The other term, object, helps users to understand the object oriented notion. Objects, also called instances of a class, are modeled on real world entities. All the instances of a class have similar properties. Basically, objects have 2 characteristics, state and behavior. State is a well-defined condition of an item which captures the relevant aspects of an object. Behavior is the observable

Figure 1. A cat object and its methods

effects of an operation or event. For example, a software object that modeled real world cat would have variables that indicated the cat's current state:

- Its breed is Cymric
- Its color is Yellow
- Its age is 3.

These variables are formally known as instance variables because they contain the state for a particular cat object, and in the object oriented terminology, a particular object is called an instance. In addition to its variables, the software cat would also have methods from real cat's behaviors: to meow, to wag the tail, play the ball etc. Figure 1 shows three different method of cat object.

A class may contain a variable that is shared by all instances of class. These variables are defined within a class. Programmer can create new objects which can inherit the number of characteristics from one of the existing objects. Objects can communicate with each other by passing message. A message is a method call from sender object to a receiver object. Object responds to a message by executing one of its methods (Legdarg, 1996). Additional information, along with the called arguments, may accompany a method call. A message includes three components. These components are object identifier which indicates the message receiver, method name, and arguments which can be required for the execution of the method. In the example of cat.play (cp), cat is an object and the receiver of the message, play is the method, and cp is the argument of play.

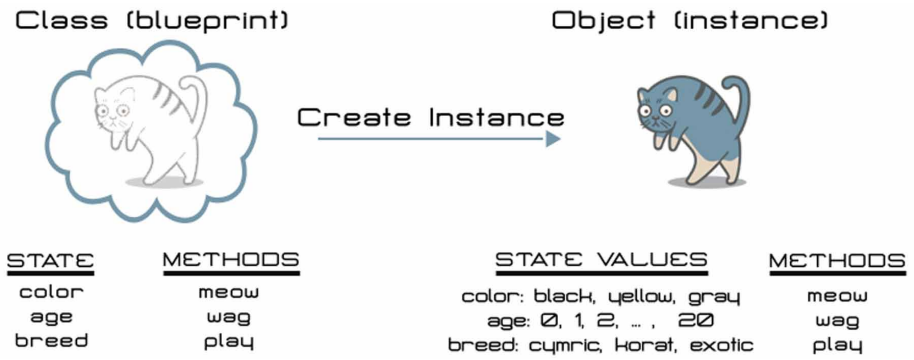
The basic implementation of 'cat' example is given in program 2 of Implementation of OOP Design in Java Programming section and this example is illustrated in Figure 2 with detail which include a diagram of cat class, cat class to address issues regarding the states, methods, attributes, etc. used in OOP.

1. OBJECT ORIENTED PROGRAMMING FEATURES

OOP has many features that make it usable including the followings:

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Figure 2. Illustration of cat object



1.1. Inheritance

This concept points code reuse without repeating or rewriting the code. It includes the creation of new classes from existing classes, and these new classes are called derived class or sub class. The existing class is called the base class or parent class or super class. The derived class reuses the base class members, moreover it can add and alter them. So programmers can avoid rewriting and testing of code that already exist. The goal of inheritance is to support refinement of classes into derived classes or subclasses. There are many advantages of using inheritance concept in programming language. Direct modeling of such hierarchies makes the conceptual structure of programs easier to comprehend, inheritance supports that common properties of classes factorized that is described once and reused when needed (Wirfs-Brock & Johnson, 1990). This results in modularity and makes complicated programs easier to comprehend and maintain, since description is avoided. Inheritance hierarchies support a technique where the most general classes containing common properties of different classes. These classes are designed and verified first, and then, specialized classes are developed by adding more details to existing classes (Thomsen, 1986).

Inheritance implements the "Is-A" relationship. In OOP, Is-A relationship means that one object is type of another one. For example, student is a person, car is a vehicle etc.

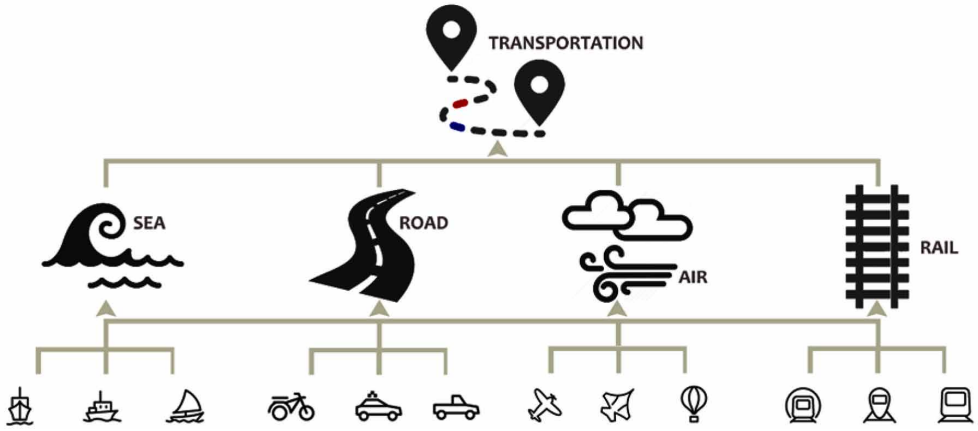
A class inherits instance variable declarations as well as methods from its base class. By adding new instance variables and new methods, and by overriding base class methods, the derived class's attributes may be redefined. A derived class cannot access the private members of its base class; otherwise this situation would be against the concept of encapsulation. A derived class has access to the public and protected members of its base class. They are both inheritable and visible to users.

There are different types of inheritance that depends on programming language. *i*) Single Inheritance: A derived class, which is inherited properties and behaviors from a single base class, is called single inheritance. *ii*) Multi-Level Inheritance: If a class is derived from another derived class, it is called multi-level inheritance. *iii*) Hierarchical Inheritance: If more than one class is derived from a base class, it is called hierarchical inheritance. *iv*) Hybrid Inheritance: It is a combine form of single and multiple inheritance. *v*) Multiple Inheritance: If a class is derived from more than one class, it is called multiple inheritance.

1.2. Polymorphism

Dictionary definition of polymorphism is the property of something having many forms. In com-

Figure 3. A real world example of inheritance



puter science, polymorphism refers to the ability of a programming language to describe objects in different ways based on their class or data type (Gamma, Helm, Johnson, & Vlissides, 1994). It can be allowed to talk to an object even if it is not known exactly what the object is. Thanks to the polymorphism, size of programming applications can be made smaller. In addition, understanding these applications is easier than those of others. If polymorphism does not exist, programmers have to check the objects one by one to determine which type and method are called according the object type. Polymorphism is activated in such situations that frees the designer from this inconvenience and allows flexibility. To better understand the concept of polymorphism, it is needed to comprehend the inheritance well. Polymorphism is tied closely to the concept of inheritance in OOP languages. There is a simple rule that is called as "is-a", to know whether or not inheritance is the right design for user's data. This rule states that every object of the subclass is an object of the superclass. To explain the relationship between inheritance and polymorphism, an example is given in Figure 3. The figure represents a transportation class with subclasses including corresponding modes (sea, road, air, rail) and vehicles. For example, all the road vehicles are a vehicle type. Thus, it makes sense for the road vehicle class to be a subclass of the vehicle class. Naturally, the opposite is not true since not every vehicle is a road vehicle. All cars and bikes are road vehicles, so they can be grouped as a subclass of road vehicle class. Race car and normal car that are not shown in the figure are indicated subclasses of car class as seen below example. The most general concept of inheritance can be explained in this way:

Common features of race car and normal car:

- They are road vehicles
- They have an engine.
- They are designed to carry persons.
- They consume the fuel.
- They need a driver.

There are common features of car and bike:

- They are road vehicle
- They are designed to carry persons.
- They need a driver.

There are common features of road vehicle and sea vehicle:

- They are designed to carry persons.
- They need a driver.

As seen in the example, information just like "to carry persons" and "to need a driver" are repeated more than one. Each category of information does not need to be saved in these classes, because inheritance provides it directly. If common features are defined in vehicle class, all subclasses can take that information from only one class. By this way, if it is necessary to update the system, only one change related to the concept in vehicle class can be enough. And this is called polymorphism.

Polymorphism is also related to "overloading" and "overriding". Overloading is a compile time polymorphism method that has the same name with different parameters. Overloading is a feature that enables a class to possess two or more methods having same name. Unless return type of method is same, error will occur.

Overriding is a run time polymorphism method that the implementation given in the base class is replaced with that in subclass. Override method can be added by rewrite the method that inherited from the base class. In this way, it provides the use of inherited class method. In this case, the software allows the flexibility that can make a different job by using the same method.

1.3. Abstraction

The word abstract means dissociated from any specific instance. Abstraction is to develop models in terms of interface and functionality instead of

implementation details. So it is related to encapsulation and data hiding (Yourdon, Whitehead, Thomann, & Oppel, 1995). Abstraction is applied to the model by considering the process of identifying object. It is used to reduce complexity of the design and implementation that focuses on the meaning of behaviors to avoid specification. Thanks to the abstraction, class internals are protected from user-level errors which breaks state of the objects (Wegner, 1987).

Abstract class is a parent class, which allows inheritance, containing abstract members. These members are only declared, not implemented. Implementation of abstract members is done within the derived class. Another type of a member is virtual member. Unlike abstract member, virtual members are implemented in parent class. To declare an abstract class, abstract keyword is used. Abstract member functions and properties are also declared with this keyword.

1.4. Encapsulation

From the user's point of view, a number of features are packaged in a capsule to form an entity. This entity offers a number of services in the form of interfaces by hiding the implementation details (Canning, Cook, Hill, & Olthoff, 1989). The encapsulation term is used to describe the hiding of the implementation details. The advantages of encapsulation are information hiding and implementation independence. Local variables are hidden in functions and private members are hidden in classes. Therefore, external direct access is prevented. If user does not know the implementation details, it is called information hiding. If user's interface is not affected by changing the implementation mechanics, it is called implementation independence (Booch, Maksimchuk, Young, Conallen & Houston, 2007). Class encapsulates the static attributes and the dynamic behaviors into the limited area to isolate and reuse when necessary. These operations cannot be done in the traditional programming languages. Private access control modifier hides data member of a class from outside world. Access to these private data members is provided via public assessor functions. Objects do not have permission to know the implementation details of others. The implementation details are hidden within the class.

2. OOP DESIGN

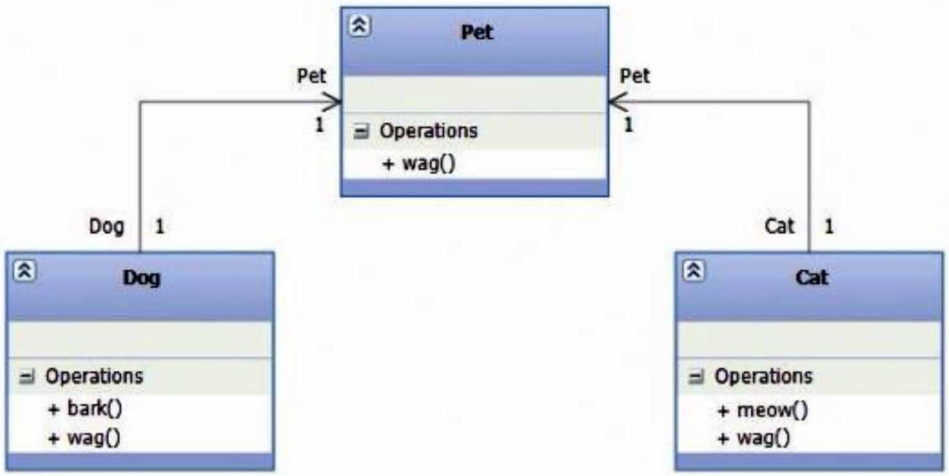
People tell the daily concept with using spoken language. Programmers try to express the concept and the entity that is related to the problem, to computer with using programming languages. To do this, during the design phase, models providing an expression are created. Object-oriented method provides the creation of these models and (if necessary) updating the system.

In section 2.1 we have developed an object-oriented design for our cat system and in section 2.2 we implemented our object-oriented design in Java and showed how to convert class diagrams to Java code.

2.1. OOP Design with the UML Diagrams

Unified Modeling Language (UML) diagrams are used to describe the structure of systems. It is a standard for modeling object-oriented systems which defines sets of rules and vocabulary for conceptual and physical representation of system (Mallick & Das, 2013). It includes graphical notation to create visual model of the system (Booch, Rumbaugh, & Jacobson, 2005) which has well-defined semantics. UML in development process is used for object-oriented analysis and design. There are different diagramming techniques which is used to model a system. These diagrams can be grouped into 2 which are called structure and behavior diagrams. Structure diagram shows the static relationships and represents the data in a system which includes class, object, package, deployment, component, composite structure and profile diagrams. Behavior diagram shows the dynamic relationships between the instances or ob-

Figure 4. Class diagram modeling generalization of superclass Pet and subclasses Cat and Dog



jects. It includes use-case, sequence, statechart and activity diagrams. First of these diagrams called use case diagram is used to capture the requirements of the system and shows the relationships between the system and environment. Sequence diagram is a form of interaction diagram which is used to model the behavior of objects by focusing on time-based ordering of an activity. Statechart diagram shows the behavior of the classes. Activity diagram illustrates the flow of activities in a use case (Booch, Rumbaugh, & Jacobson, 2005).

It has developed an object-oriented design with UML class diagram for pets as shown in Figure 4.

Figure 4 is the class diagram that models generalization of superclass Pet and subclasses Cat and Dog. The following section 2.2, it is implemented our object-oriented design in Java and showed how to convert class diagrams to Java code (Deitel & Deitel, 2012).

2.2. Implementation of OOP Design in Java Programming

A university system can be given as an example, and this system is implemented by java programming language. The Java source code for the Person class is shown in Listing 1.

In this example, there is a base class which is called Person. This class includes basic information, which people have, and there are 3 derived classes which inherit data and functions from the base class. These derived classes have their own methods and data members, and also they have methods and data members of Person class.

Keyword public is an accessibility modifier which refers the calling members or methods from external locations like other classes (Lea, 1999). AcademicPersonnel, administrativePersonnel and student classes call methods from Person class. Derived class methods are declared public because, they should be accessed from Main() method which Test class has. AcademicPersonnel class has a method called checkAcademicPersonnel(), this method is declared private. Private access specifier refers hiding data members and methods from other classes. Every class can access its own private data members and methods. If we try to reach this private checkAcademicPersonnel() method from Test class, it is going to fail. deliverCourse() method, which is declared public, can reach this private method because, they are in the same class.

Last class is called Test, which is developed to test other classes, it only has one method called Main. We create 3 different objects from 3 different derived class. Object academicPersonnel1 is created from academicPersonnel class which has deliverCourse() method on its own scope but, this

Listing 1.

```
1. public class Person
2. {
3.  public void printPersonalInformation(String id, String fName, String lName)
4.  {
5.  System.out.println(id+" "+fName+" "+lName);
6.  }
7.  }
8.  class academicPersonnel extends Person
9.  {
10.  private boolean checkAcademicPersonnel()
11.  {
12.  return true;
13.  }
14.  public void deliverCourse(String courseCode, String courseName)
15.  {
16.  if(checkAcademicPersonnel())
17.  {
18.  System.out.println("Academic Personnel");
19.  System.out.println("Course: "+courseCode+ " "+ courseName);
20.  }
21.  }
22.  }
23.  class student extends Person
24.  {
25.  void takeCourse(String courseCode, String courseName)
26.  {
27.  System.out.println("Course: "+courseCode+" "+courseName
28.  }
29.  }
30.
31.  class administrativePersonnel extends Person
32.  {
33.  boolean checkStudentInformation()
34.  {
35.  return true;
36.  }
37.  }
38.  class Test
39.  {
40.  public static void main(String[] args)
41.  {
42.  academicPersonnel academicPersonnel1 = new academicPersonnel();
```

continued on following page

Listing 1. Continued

```
43.  student student1 = new student();
44.  administrativePersonnel administrativePersonnel1 = new  administrativeP-
ersonnel();
45.  academicPersonnel1.printPersonalInformation(("100305044","Rahime","Yilm
az");
46.  academicPersonnel1.deliverCourse("CENG101", "Algorithms and Programming
I");
47.  student1.printPersonalInformation(("214700560", "Anil", "Sezgin");
48.  student1.takeCourse("CENG102", "Algorithms and Programming II");
49.  administrativePersonnel1.checkStudentInformation();
50.  }
51.  }
```

object can also reach printPersonalInformation() method on Person class. Object student1 is created from student class, which has takeCourse() method on its own scope, and same as academicPersonnel class, student class is derived from Person class, so this object can reach base class methods. Third object is created from administrativePersonnel class which is derived from same base class.

For another example of the Java source code for the pet class is shown in Listing 2.

This example shows using abstract class. Abstract class is used to create a base template for derived classes. Pet class is the abstract base class which includes an abstract method called wag(). This method has no implementation, therefore it is declared abstract. There are two classes called dog and cat which are derived from pet class. These two derived classes provide an override method by using override keyword. Cat and Dog classes override wag() method, dog1.wag() invokes the wag() method declared in dog class, cat1.wag() invokes the wag() method declared in cat class.

FUTURE RESEARCH DIRECTIONS

To create (write) a program in any languages needs detailed training. That means a programmer have to be trained highly in the programming language that programmers wants to use. In the real life, even if any person do not know any languages, they could write a computer program. Namely, untrained people can also create the program without using any programming languages. We can ask how it can be done. Instead of teaching a programming languages to a person, we can teach how to draw flow charts. When a correct flow chart is drawn, a case at the background of the flow chart can create a program source code by using the flow chart. If these kinds of cases can be created, an untrained person can write his program by using flow chart and produce his programming code. First of this, the basic flow chart symbols are taught and combination of this basic elements are told to the training people who wants to implement their problem by using flow chart. And then, it is expected to solve their problem. This solves the dependency of a programming language problem to implement a computer software.

CONCLUSION

The programming languages before OOP concept were not easy and friendly. Large and complex problems were solved by dividing into small systems. The real world modeling was not implemented by procedural programming. Creation of the real simulation of problems can be implemented

Listing 2.

```
1.  public abstract class pet
2.  {
3.  public abstract void wag();
4.  }
5.  class cat extends pet
6.  {
7.  @Override
8.  public void wag()
9.  {
10.  System.out.println("Cat wags tail.");
11.  }
12.  }
13.  class dog extends pet
14.  {
15.  @Override
16.  public void wag()
17.  {
18.  System.out.println("Dog wags tail.");
19.  }
20.  }
21.  class Test1
22.  {
23.  public static void main(String[] args)
24.  {
25.  dog dog1 = new dog();
26.  cat cat1 = new cat();
27.  dog1.wag();
28.  cat1.wag();
29.  }
30.  }
```

by the OOP. The functional programming shows more tendency for losing data while running according to OOP. The system requirements of data structure and data are more flexible in OOP than those in other models. The modification of all systems is more complicated than the object oriented system, since OOP system has a modular structure. All processes are done by using functions in former languages but in OOP, everything is processed by using objects methods. The data is transferred by using messages among objects. By this way, objects can communicate among themselves. Data hiding properties of OOP attempt to protect the data from the outside modification request.

As a conclusion, the features of OOP, such as encapsulation, abstraction, polymorphism and inheritance help us to model the real life entities on computers. A more qualified and effective software can be created by using OOP.

REFERENCES

Bista, R., Bajracharya, L., & Dongol, D. (2015). A New Approach To Enhance Efficiency of Object Oriented Programming. *Technia*, *8*(1), 1058.

Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Conallen, J., & Houston, K. A. (2007). *Object-Oriented Analysis and Design with Applications* (3rd ed.). Boston, MA: Addison-Wesley Professional.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005, May). *The Unified Modeling Language User Guide.* Addison-Wesley.

Canning, P. S., Cook, W. R., Hill, W. L., & Olthoff, W. G. (1989). *Interfaces for Strongly-Typed Object-Oriented Programming*. Paper presented at OOPSLA '89, New York, NY. doi:10.1145/74877.74924

Deitel, P., & Deitel, H. (2012). *Java How to Program* (9th ed.). Pearson Education Limited.

Dennis, A., Wixom, B. H., & Tegarden, D. (2015, April). *System Analysis & Design: An Object-Oriented Approach with UML* (5th ed.). Academic Press.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley Professional.

Harel, D., Marron, A., & Weiss, G. (2010, June). Programming coordinated behavior in java. In *European Conference on Object-Oriented Programming* (pp. 250-274). Springer Berlin Heidelberg.

Lea, D. (1999). *Concurrent Programming in Java: Design Principles and Patterns*. Boston, MA: Addison-Wesley Professional.

Ledgard, H. F. (1996). *The Little Book of Object-Oriented Programming*. Upper Saddle River, NJ: Prentice Hall.

Mallick, B., & Das, N. (2013, November). An Approach to Extended Class Diagram Model of UML for Object Oriented Software Design. *International Journal of Innovative Technology & Adaptive Management, 1*.

Robson, D. (1981). Object-Oriented Software Systems. *Byte.*, *6*(8), 74–86.

Thomsen K., S. (1986). *Multiple Inheritance, a Structuring Mechanism for Data, Processes and Procedures*. Aarhus Universitet, Matematisk Institut, Datalogisk Afdeling.

Wegner, P. (1987). *Dimensions of Object-Based Language Design.* Paper presented at OOPSLA '87, New York, NY. doi:10.1145/38765.38823

Wirfs-Brock, R. J., & Johnson, R. E. (1990). Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33.

Yourdon, E., Whitehead, K., Thomann, J., & Oppel, K. (1995). *Mainstream Objects: An Analysis and Design Approach for Business* (1st ed.). Upper Saddle River, NJ: Prentice Hall.

KEY TERMS AND DEFINITIONS

Class: Used to define the objects and to describe the contents of the objects.

Java: A high level programming language that is based on object oriented design which was first released by Sun Microsystems in 1995.

Message: Communication type among methods.

Method: Called a function or a procedure that is used to operate on the data.

Object: Instance occurrence of a same class that models real-world items.

Object Oriented: A technique for programming that is based on the objects and on the relationship between those objects.

Programming: A mathematical methodology that helps programmers while writing a program.