Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

OOP For Noobs

Introduction

Hello there my fellow enthusiastic students. I have created this document in order to aid you in your journey of understanding Java, and by extension… Object Oriented Programming Concepts. I wish you all, nothing but the best going forward in your endeavors.

Overview

This “book” will cover most of the fundamentals of Object-Oriented Programming (OOP) concepts that we will cover in the Java Fundamentals Course. As Java is a language designed around OOP principles, by having a fundamental understanding of OOP Concepts, you will be quickly able to master Java and start building applications.

My goal for this document is to present the information in a way that can be interpreted easily by you as a student.

First, we will take a look at what Java is exactly, and why it’s actually worth learning. After a brief overview, we will then focus entirely on OOP principles, and getting you acquainted with the various terms and concepts.

I’ve also decided to include summaries and TLDRs (Too long Didn’t Read) for the lengthier parts just in case.

In each section, I will try to provide adequate examples and exercises to assist. My email is ~~NOT~~ in the header, so if you have any questions or comments, feel free to let me know.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Contents

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Overview of Java

Java is a high-level, object-oriented programming language that is designed to be platform-independent, meaning it can run on any computer or operating system that has a Java Virtual Machine (JVM) installed. Java was first released in 1995 by Sun Microsystems and is now owned by Oracle Corporation.

One of the key features of Java is its "write once, run anywhere" philosophy, which means that developers can write Java code on one platform and run it on any other platform without having to recompile it. This is made possible by the use of the JVM, which interprets Java bytecode and executes it on the host platform.

Java is widely used for developing enterprise applications, web applications, mobile applications, and games. It is also popular in academia for teaching computer science and programming concepts. Some of the main features of Java include:

Object-oriented programming

Garbage collection

Platform independence

Exception handling

Multi-threading

Generics

Annotations

Java is also known for its large standard library, which provides a wide range of classes and interfaces for developers to use in their applications. The standard library includes classes for networking, file I/O, XML parsing, graphics, user interface development, and more.

Overall, Java is a versatile and powerful programming language that is used in a wide range of applications and industries.

TLDR;

It's a really cool language with a lot of applications and potential in a wide range of fields. Because of how old it is as well, it also has a large community and various resources to help you get started and make applications and projects.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Object Oriented Programming Concepts

Overview

Object-Oriented Programming (OOP) is a programming paradigm that is centered around the idea of objects that encapsulate data and behavior. OOP aims to model real-world entities in software design by providing a framework for creating classes that represent these entities.

The main concepts of OOP include:

- Abstraction: Hiding the implementation details and providing only the necessary details to the user. It allows us to focus on what the object does instead of how it does it.

- Encapsulation: Enclosing the data and code within an object to prevent direct access by the user. It helps in achieving data hiding and improves maintainability.

- Inheritance: A mechanism by which one class can inherit properties and behavior from another class. It is used to create a hierarchical classification of objects and to reuse code.

- Polymorphism: The ability of an object to take on many forms. In Java, it can be achieved through method overloading and method overriding.

These concepts provide a powerful framework for creating flexible, modular, and maintainable software.

We'll go over each of these four (4) concepts in details with examples to ensure that you understand these concepts.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Classes & Objects

Before we can start looking at some of the concepts behind OOP, we have to first really understand how it works conceptually.

The building blocks of OOP are Classes, and Objects. Unless you understand these two (2) terms and what they are, everything else becomes difficult.

What is a Class?

As we can recall, Object Oriented Programming is designed to model our real world. As such, we can explain each of the concepts using Real World Examples.

A Class can be thought of as a template, or a blueprint. For this example, we will consider a blueprint of a house. This blueprint of the house will explain the various properties and functionalities of the house.

For an example focusing on properties… We know that each house will have a color, size, number of doors, number of rooms, etc. Each of these are things that tell us about what the object should have, or even better, they describe the object to us. Another example would be a Human. Some properties of a human such as yourself, (assuming you're a human), would be characteristics such as your skin color, hair color, number of hands, personality etc. These are all things that general humans will have.

Figure 1: Image showing a blueprint of a house

Now assuming that we all have a general understanding of properties, let's take a look at behaviours. The behaviours of a class tell us what the class is capable of doing. Using the human class as an example, certain things the human is capable of doing include: walking, talking, moving, thinking, growing, and much more.

Review:

A class is like a template, which contains properties and behaviours.

These properties describe the class, and the behaviours tell us what the class is capable of doing.

What is an Object?

You've probably heard that an object is an instance of a class, but never quite understood what that really means. So let's break it down.

Remember, our class is a template, which we can use to create several copies of it. For example, you have your blueprint of a house, you can create several houses, while following that blueprint. Each house can also have different values for their properties. By this, I mean that while each house has the same properties as the template, that being the color, size, etc, the exact values for each of these houses can differ. Using the same blueprint, we can create a blue house, a red house, a purple house, a bigger house, a smaller house, etc etc. Same thing goes for humans. We have our generic human blueprint, which has various properties and behaviours, and we can "create" various humans with slightly different values for their properties. For example, black humans, white humans, humans with 10 fingers, humans with none, and so on.

Figure 2: Example of a house created from a blueprint

Each copy that we make using that blueprint, is an object. When we say that an object is an instance of a class, we are saying that we took our blueprint, and we made an instance of it in “reality”.

Final example for now, we know that a computer must do certain things and have certain properties in order for it to be considered a computer. Yet, our computers that we use can be vastly different, but they are still computers. They’re all made from the same template that defines the core behaviour and attributes, but can be modified upon to get different results.

Review:

Objects are “physical” copies of a blueprint that we can create, and give various values to.

Classes define the structure for how an Object should be described and behave, and Objects take that structure, and use it to become “real” or exist.

Exercise:

Only the first part and there’s already a lot of information, so let’s do a few exercises to make sure everyone is still following.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

I want you all to look around you, and realize how everything that you can see can be thought of as an object. Even the device that you are reading this on is an object. Anyway, look around and find 5 objects, and write down their properties (how they can be described) and their behaviours (what they can do). Make sure you WRITE/TYPE IT DOWN! This information will not stick unless you start to actually understand what's going on. Go on, start looking and writing!

Conclusion:

I pray you're reading this AFTER completing the above exercise! In this section, we took a basic look at the core building blocks of Object Oriented Programming, which are Classes and Objects. We looked at what a class is, and how they can be used to create objects, and what objects are, and how they can be created by a class. You wouldn't be confused if you did that exercise! So make sure you do it.

Next, we'll start taking a look at how these building blocks connect with other concepts to create Object Oriented Programming.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Abstraction

So, keeping it nice and simple, Abstraction is one of the four main concepts of Object-Oriented Programming that we will be going through.

What is it?

Abstraction is one of the fundamental principles of object-oriented programming. It refers to the process of hiding implementation details while showing only the necessary information to the user. In simpler terms, abstraction means focusing on the essential features of an object or system while ignoring the rest. It a

Example:

A common example of abstraction is a car. When you drive a car, you don't need to know how the engine works, how the transmission works, or how the fuel injection system works. All you need to know is how to use the steering wheel, pedals, and gear shifter to operate the car. The implementation details of the car's inner workings are hidden from the driver, allowing them to focus on the essential features.

Advantages & Disadvantages

Advantages of Abstraction	Disadvantages of Abstraction
Encourages code reusability and reduces code duplication	Can make code more complex and harder to understand
Makes code more flexible and easier to modify	Can sometimes reduce performance due to additional layers of abstraction
Allows for better organization and management of code	Can lead to over-abstraction and unnecessary complexity
Helps to reduce dependencies between modules and components	Requires more time and effort to design and implement
Improves maintainability by separating interface from implementation	Requires a good understanding of the underlying problem domain

It's important to note that while there are some potential disadvantages to abstraction, the benefits generally outweigh the drawbacks. When used appropriately, abstraction can help to improve the quality, reliability, and maintainability of software systems.

Conclusion:

Abstraction allows us to have an object, and hide the details from the end user. Take your smart phone for example, you most likely have no clue how your phone actually works, or what materials were used to make it, but you do know HOW to use it, which is what’s important and often all you need to know as an end user.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Encapsulation:

What is it?

Encapsulation is one of the four fundamental principles of Object-Oriented Programming (OOP) that aims to achieve data hiding and protect the internal state of an object from external access and modification. It is a mechanism of binding data and code that manipulate it, such that they can be hidden from the outside world.

Example

Imagine a company that has a payroll system to calculate employee salaries. The payroll system has various methods and properties that should only be accessible to authorized personnel. To implement encapsulation in this scenario, the company can create an object-oriented design where the payroll system is encapsulated in a class that restricts access to its methods and properties.

For example, the class could have a private property for the current balance of the payroll system, which can only be modified by authorized personnel through a setBalance method that includes authentication checks. Similarly, the class could have a public method for calculating an employee's salary, but this method would only be accessible to authorized personnel who have been granted access through a login process.

This way, the sensitive information and functionality of the payroll system are encapsulated in the class, making it more secure and less prone to errors caused by unauthorized access or modification.

Benefits:

Encapsulation provides several benefits, including:

Data hiding: Encapsulation ensures that the internal state of an object is hidden from the outside world, which prevents accidental modification of data by external users.

Increased security: Encapsulation makes it difficult for hackers to access and modify data, which can improve the security of an application.

Improved maintainability: Encapsulation simplifies the modification of a class without affecting the code that uses the class.

Reusability: Encapsulation improves the reusability of code by making it possible to use existing classes without worrying about the internal details of those classes.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

However, there are also some potential drawbacks to encapsulation, such as:

Increased complexity: Encapsulation can make the code more complex, as it requires the creation of additional methods to access and modify the data.

Reduced performance: Encapsulation can sometimes reduce performance, as it requires additional method calls to access and modify data.

Conclusion / TLDR:

In conclusion, encapsulation is an important principle of object-oriented programming that helps to keep data and methods hidden from unauthorized access. It ensures that the internal workings of an object are not exposed to the outside world, and that all access to the object's data and behavior is performed through well-defined interfaces. Encapsulation also helps to reduce complexity, increase modularity, and improve maintainability of code. By using encapsulation, we can create more robust and secure software systems that are easier to understand and maintain over time.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Inheritance:

Now it’s big boy time.

Inheritance is a key concept in object-oriented programming that allows a new class to be based on an existing class. Inheritance allows the new class to inherit the properties and behaviors of the existing class, which can save time and effort in development by reusing code that has already been written.

An example of inheritance can be seen in the animal kingdom. Consider the class of animals called "mammals." Mammals have certain characteristics in common, such as the ability to nurse their young and having hair or fur. Within the class of mammals, there are sub-classes, such as primates, canines, and felines. These sub-classes inherit the characteristics of mammals but also have their own unique characteristics. For example, primates have opposable thumbs, canines have sharp teeth, and felines are known for their agility.

In this example, the class of mammals is the parent or base class, and the sub-classes are the child or derived classes. The child classes inherit the properties and behaviors of the parent class but can also add their own unique properties and behaviors. Similarly, in object-oriented programming, a child class can inherit the properties and methods of a parent class and add its own unique properties and methods.

That’s all it is, not so scary now is it.

Conclusion / TLDR

Inheritance is a key concept in object-oriented programming that allows a subclass to inherit properties and methods from a superclass. This makes it easier to reuse code and create more specialized classes. Inheritance is often used to create class hierarchies, where subclasses become more specific versions of their parent classes. An example of inheritance could be a family tree, where a child inherits certain physical and personality traits from their parents, who may have inherited those traits from their own parents, and so on. Inheritance allows for the creation of more complex and nuanced relationships between classes, making object-oriented programming more flexible and powerful.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Polymorphism:

Polymorphism is the ability of an object to take on multiple forms or behaviors. In object-oriented programming, it refers to the ability of objects of different classes to be used interchangeably, as long as they implement the same interface or inherit from the same superclass.

There are two main types of polymorphism:

Compile-time polymorphism, also known as method overloading, where methods with the same name but different parameters can be defined in a class.

Runtime polymorphism, also known as method overriding, where a subclass provides a specific implementation of a method that is already provided by its parent class.

An example of polymorphism can be seen in a shape hierarchy. Suppose we have a superclass called "Shape" and subclasses called "Circle," "Rectangle," and "Triangle." Each subclass would inherit properties and methods from the superclass, but could also have its own unique properties and methods. For example, each shape could have a method called "area," but the implementation of that method would be different for each shape. This allows us to write code that can work with any shape object, without needing to know the specific type of shape.

Conclusion / TLDR:

Polymorphism is the ability of an object to take many forms. In Java, polymorphism is achieved through method overriding and method overloading. Method overriding is when a subclass provides its own implementation of a method that is already present in its parent class, while method overloading is when multiple methods have the same name but different parameters. Polymorphism allows for more flexible and extensible code, as well as the ability to write code that can handle multiple object types without knowing their specific class at compile time.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

BONUS!

Oh? You made it to the end? Congratulations! For being such a good student, we'll look at one more thing… Composition!

Composition is another object-oriented programming concept that allows one object to contain another object. It is a way to create complex objects by combining simpler ones. In composition, an object is made up of one or more other objects, and the component objects cannot exist independently of the composite object.

Composition is often used to represent complex relationships between objects where one object contains or is part of another object. For example, a car can be composed of an engine, a transmission, wheels, and other parts. The car object can then use the methods and properties of its component objects to perform its functions.

Composition is different from inheritance, where a subclass inherits the properties and methods of its parent class. With composition, an object can contain objects of different classes and can delegate tasks to these objects as needed.

Composition can help make code more modular and flexible, allowing developers to create complex systems with smaller, more manageable components. It also supports the principle of encapsulation by allowing objects to hide their internal implementation details.

Conclusion / TLDR:

Composition is a design technique used in object-oriented programming where objects are created by combining different smaller objects, rather than inheriting properties and behavior from a parent object. This allows for greater flexibility in creating complex objects, and allows objects to be reused in different contexts. Composition helps to reduce complexity, improve code organization, and maintainability.

Not Isaiah Carrington
Not Isaiah.carrington@mycavehill.uwi.edu

Summary:

Thanks for making it all the way to the end. By this point, assuming I did a good job, you must be having a sense of sadness at not having more information by way of Java examples. Genuinely, I was going to put them in, but I decided that I didn't want to confuse you with the syntax, and would rather just describe the theory and principles. If you REALLYYYY want one with code examples and stuff, taking a deeper yet general and easily digestible look at Java, let me know!

Thank you for your time.

Signed:

Not Isaiah Carrington 😎