+33 (0)1 48 10 09 46
formation@ambient-it.net
140 Avenue Jean Lolive 93500 Pantin - France

Updated 06/06/2025

Sign up

Training in Object-Oriented Programming (OOP)

3 days (21 hours)

Presentation

Fun, modern and based on real-life cases, our Object-Oriented Programming course is designed to guide you in mastering the fundamental concepts of OOP, while integrating artificial intelligence (AI) as an everyday ally. The main objective is to enable you to move from an old-fashioned functional and procedural approach to a modern object-oriented one, by understanding its key principles: modularity, code reuse, extensibility and its adaptation to the real world.

During this course, you'll discover the benefits of OOP and AI in your code, the differences between recent languages such as C++, Python, Java and C#, and how this paradigm addresses your problems as a developer. Through our unique pedagogy, you'll easily learn and retain the main principles of OOP, such as abstraction, encapsulation, inheritance and polymorphism.

One of the highlights of this training course is the integration of AI into your daily life as a developer. You'll learn how to use AI to generate code, improve its design, refactor it and document your projects more efficiently. Through hands-on workshops, you'll apply your skills to business use cases, using inheritance and polymorphism, while getting help from generative AI tools to code faster and smarter.

By the end of this course, you'll be ready to design flexible, scalable and maintainable applications while harnessing the potential of AI to improve your productivity.

Objectives

- Understand the principles and specifics of object-oriented programming
- Moving from a functional to an object-oriented approach
- Discover the impact of AI on object-oriented programming
- Implement a simple project integrating object-oriented programming and AI

Target audience

- Developers
- Analysts
- Project managers wishing to move into object-oriented development technologies

Prerequisites

- Basic knowledge and experience in application design and software development.

Object Oriented Programming training program

Programming paradigms: from procedural to object-oriented

- What is a Paradigm?
- History and evolution of OOP
- The main object-oriented languages and their main differences: C++, Python, Java, C#
- Understand the fundamental differences between procedural (imperative) and object-oriented programming
- Identify the limits of a traditional functional approach in modern applications (maintainability, upgradeability)
- Discover the key advantages of OOP: modularity, code reuse, extensibility and a better match with the real world.
- Familiarize yourself with basic object terminology (class, instance, method, attribute) and the OOP vocabulary versus procedural vocabulary
- Practical workshop: Case study - Based on a simple scenario (e.g. managing a bank account), group discussion of a procedural versus an object-oriented solution. Learners identify the objects and responsibilities in the problem and sketch out how to organize them into classes.

The Art of Prompt: Making AI an everyday ally of OOP

- Formulate useful prompts for generating or improving object-oriented code
- Use AI as a design, refactoring or documentation assistant
- Gain clarity, productivity and creativity with the right dialogue reflexes
- Discover integrated tools (Cursor, Windsurf, etc.) to code with AI on a daily basis.
- Hands-on workshop with the help of generative AI: Write a prompt to generate two linked classes, where a customer can have several accounts.

Fundamental OOP concepts: classes, constructors, objects and methods

- Assimilate the concept of class (model) versus object (concrete instance) and understand the relationship between the two
- The builders and the destroyer
- Attributes and methods for structuring data
- Distinguish between instance attributes (specific to each object) and class attributes (static/shared) and understand the usefulness of each
- Object lifecycle management
- Practical workshop: Creation of a simple class, the CompteBancaire class, with a few attributes (holder, balance) and methods (deposit, withdraw, display balance).

Demystifying the 4 main principles of OOP

- Abstraction: simplifying complexity by hiding detail
- Encapsulation: data grouping and associated methods
- Inheritance: reuse and specialize existing classes
- Polymorphism: managing different implementations under a common interface
- Introduction to UML diagrams for visualizing class hierarchies and relationships
- Practical workshop with the help of generative AI: Reuse your BankAccount class and add to your project classes CurrentAccount and SavingsAccount, using the 4 principles of OOP.

Visibility and Encapsulation

- The importance of encapsulation to protect data
- Visibility differences between public, private and protected
- The impact of visibility on safety and maintainability
- Getter/setter access methods
- Practical workshop: Create a secure class with private attributes and controlled access via getters/setters. Protect member data (balance, owner, etc.) by declaring them private and providing accessors (getSolde(), getPropriétaire()) and mutators (déposer(), retirer()).

Inheritance: factoring and extending classes

- Understand the principle of inheritance in OOP: create new classes by extending an existing class to reuse common code
- Differentiate between parent class (superclass) and derived class (subclass); understand how to a subclass inherits the attributes/methods of the superclass
- Implement specialization through inheritance: add or redefine (override) methods in a subclass to adapt inherited behavior to specific needs
- Understand the advantages (centralized common factor, consistency) and risks of inheritance (strong coupling, complexity) in order to use it wisely
- Practical workshop with the help of generative AI: Extending an existing class. Taking the BankAccount class, participants design a SavingsAccount subclass that inherits from BankAccount. For example, they add an interest rate attribute and redefine the interest calculation method. Next, they write a mini usage scenario showing how a classic SavingsAccount and BankAccount can be polymorphically manipulated via their common parent class.

Polymorphism and abstraction: towards flexible design

- Understanding polymorphism (using the same interface with several classes)
- Distinguishing between abstract classes and interfaces
- Design generic code (reusable and upgradeable)
- Discover the principle: "program against an interface, not an implementation".
- Practical workshop: Creation of a CompteBancaire interface grouping methods common to different account types (deposit, withdrawal, balance display). Then implement this interface in specialized classes (CompteCourant, CompteÉpargne) to illustrate how polymorphism facilitates uniform management of varied objects. This exercise will clearly demonstrate the value of abstractions for designing flexible, easily maintainable code.

Concepts extended to the 4 principles of OOP

- Composition: create complex objects by combining simpler ones ("has one" relationship)
- Cohesion: Ensuring that a class or method has a single, clearly defined responsibility
- Weak coupling (or Decoupling): Minimize dependencies between objects or components to facilitate maintenance and scalability.
- Practical workshop: Create a BankCard class associated with each account ("has one" relationship). card").

The Code is your Kingdom: Introduction to the SOLID principles

- Refactoring techniques: How do you transform spaghetti code into clear code, organized into coherent classes?
  - extract method
  - extract class
  - rename
- Identifying warning signals
- S - Single responsibility: one class = one mission, like a specialized knight.
- O - Open/Closed: your code accepts change... without being sabotaged.
- L - Liskov substitution: heirs must respect the inheritance (no betrayal!).
- I - Interface segregation: better several small tools than one big useless gadget.
- D - Injecting dependencies: entrusting the keys to the outside: the class remains free, flexible and testable.
- Practical workshop: Critical analysis and rewriting of a code according to SOLID principles, in order to make it more effective.
  improve maintainability by refactoring its code.

Design Patterns: The developer's secret recipes

- Importance and benefits of Design Patterns
- Design patterns (Factory, Singleton, etc.)
- Structuring patterns (Adapter, Facade, etc.)
- Behavior patterns (Observer, Strategy, etc.)
- Practical workshop: Keep an eye on your bank accounts with the Observer to avoid being overdrawn!

Conclusion and opening to Software Architectures

- MVC architecture
- Introduction to Microservices: fundamental principles, benefits
- Service-oriented architecture decoupling and modularity
- Practical workshop with the help of generative AI: ask the AI to produce a skeleton of MVC architecture code.

Further information

Companies concerned

This course is aimed at both individuals and companies, large or small, wishing to train their teams in a new advanced computer technology, or to acquire specific business knowledge or modern methods.

Positioning on entry to training

Positioning at the start of training complies with Qualiopi quality criteria. As soon as registration is finalized, the learner receives a self-assessment questionnaire which enables us to assess his or her estimated level of proficiency in different types of technology, as well as his or her expectations and personal objectives for the training to come, within the limits imposed by the selected format. This questionnaire also enables us to anticipate any connection or security difficulties within the company (intra-company or virtual classroom) which could be problematic for the follow-up and smooth running of the training session.

Teaching methods

Practical course: 60% Practical, 40% Theory. Training material distributed in digital format to all participants.

Organization

The course alternates theoretical input from the trainer, supported by examples, with brainstorming sessions and group work.

Validation

At the end of the session, a multiple-choice questionnaire verifies the correct acquisition of skills.

Sanction

A certificate will be issued to each trainee who completes the course.