BCS LEVEL 5 DIPLOMA IN IT

OBJECT-ORIENTED PROGRAMMING

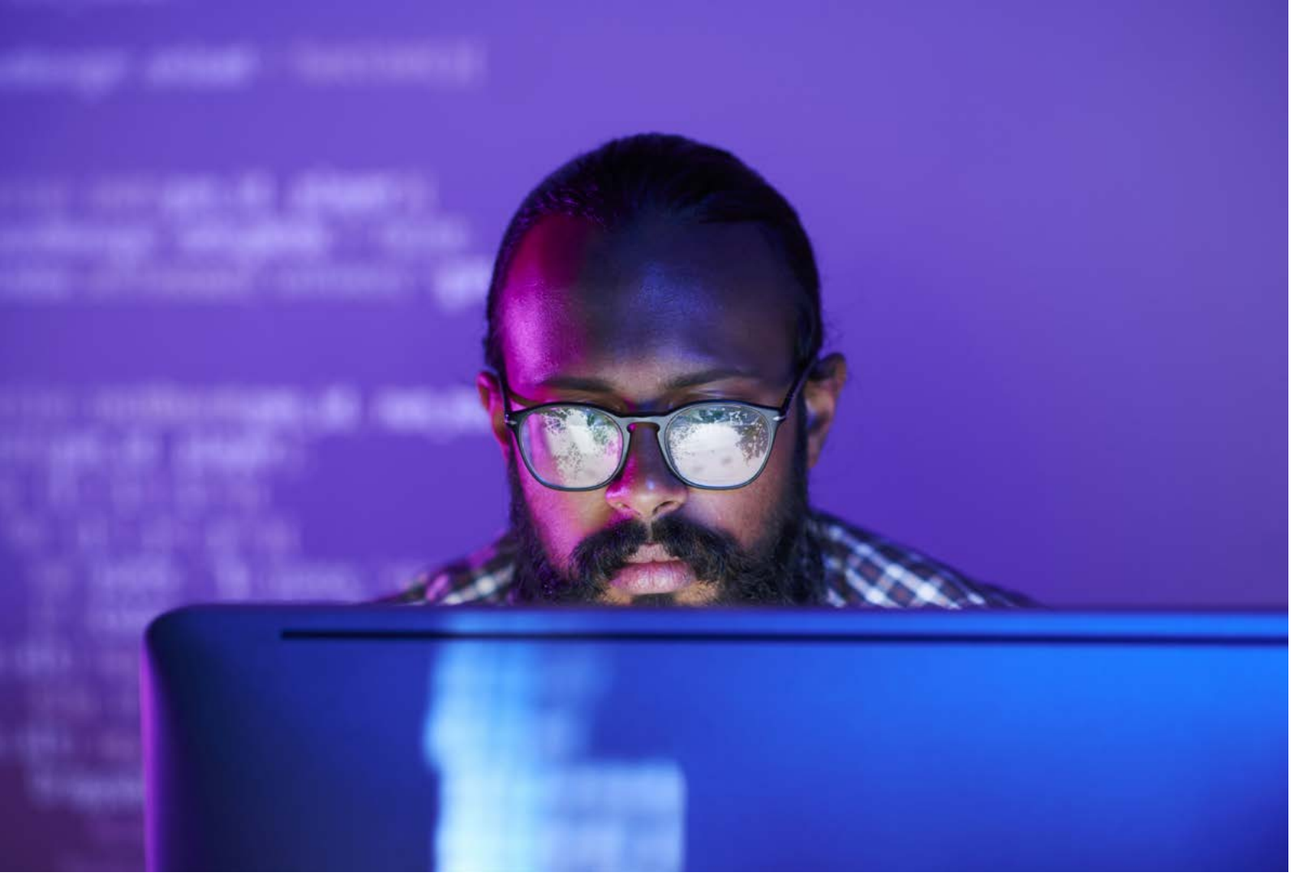
SYLLABUS

THIS QUALIFICATION WILL BE RETIRING IN 2026

September 2023 v4.1

bcs The Chartered Institute for IT

CONTENTS



Introduction

Level 5 Diploma in IT

The second stage within the BCS three-stage Higher Education Qualification programme, the Level 5 Diploma enables candidates who have already achieved the Level 4 Certificate in IT to progress to higher levels of knowledge and competency.

This internationally-recognised qualification introduces you to the business-related aspects of the IT industry, developing your technological expertise while also considering the potential challenges of the day-to-day running of an organisation, such as legal obligations and intellectual property.

Our modules have been created in-line with the latest developments in the industry, giving you a competitive edge in the IT job market. You will have the opportunity to learn about object-oriented programming, user experience, systems analysis and design, as well as to build upon knowledge and skills developed during the Level 4 Certificate.

To successfully achieve the qualification, candidates need to complete:

- One core module
- Three optional modules
- One Professional Project in IT

Candidates who wish to progress onto the next stage will need to complete the Project at end of the Level 6 Professional Graduate Diploma in IT.

Object-Oriented Programming Optional Module

The Object-Oriented Programming module is an optional module that forms part of the Level 5 Diploma in IT – the second stage within the BCS three-stage Higher Education Qualification programme.

Candidates will explore the foundations of Object-Oriented Programming, the concepts relating to abstraction, encapsulation and inheritance, as well as design patterns, Unified Modelling Language (UML) and Object Constraint Language (OCL). Candidates will have the opportunity to develop knowledge of the SOLID design principles, class hierarchies, implementation of designs and testing OO code.

Qualification Suitability and Overview

Candidates must have achieved the Certificate in IT or have an appropriate exemption to be entered for the Diploma in IT. Candidates can study for this diploma by attending a training course provided by a BCS accredited Training Provider or through self-study, although it is strongly recommended that all candidates register with an approved centre. Studying with an approved centre will deliver significant benefits.

Candidates are required to become a member of BCS, The Chartered Institute for IT, to sit and be awarded the qualifications. Candidates may apply for a four-year student membership that will support them throughout their studies.

The Level 5 Diploma is suitable for professionals wishing to gain a formal IT qualification, and this module may be particularly relevant for candidates interested in career opportunities such as software engineering, creating apps, or designing systems.

Total Qualification Time	Guided Learning Hours	Assessment Time
1086 hours	**225 hours**	**2 hours**

SFIA Levels

This module provides candidates with the level of knowledge highlighted within the table, enabling candidates to develop the skills to operate successfully at the levels of responsibility indicated.

Level	Levels of Knowledge	Levels of Skill and Responsibility (SFIA)
K7		Set strategy, inspire and mobilise
K6	Evaluate	Initiate and influence
K5	Synthesise	Ensure and advise
K4	**Analyse**	**Enable**
K3	**Apply**	**Apply**
K2	**Understand**	**Assist**
K1	**Remember**	**Follow**

SFIA Plus

This syllabus has been linked to the SFIA knowledge skills and behaviours required at Level 5.

PROG3

Designs, codes, verifies, tests, documents, amends and refactors moderately complex programs/ scripts. Applies agreed standards and tools, to achieve a well-engineered result. Collaborates in reviews of work with others as appropriate.

Further detail regarding the SFIA Levels can be found at www.bcs.org/levels.

Learning Outcomes

Upon completion of this module, candidates will be able to:

- Explain the motivation for and development of object-oriented programming languages.
- Produce a set of use cases given a problem statement.
- Produce class diagrams, object interaction diagrams and object state transition diagrams for a given problem.
- Describe the essential features of an object-oriented programming language.
- Produce and/or debug code fragments that illustrate principles of object-oriented software development.
- Describe the principles for testing object-oriented software and derive sets of test data given a specification.

Syllabus

1. Foundations

Learners will be able to:

1.1 Explain the genealogy of object-oriented (OO) languages.

Indicative content

a. Structured programming
b. Procedural programming
c. Abstract data types (ADTs)

Guidance

Candidates should be able to define these terms and understand their contribution to the development of OO languages.

1.2 Explain the difference between typed and untyped languages.

Indicative content

a. Type of variable and range of applicable operations
b. Use of classes in object-oriented programming

Guidance

The syllabus is designed to be language-agnostic, although it leans towards languages with classes rather than those without. Candidates are expected to be proficient in one OO language and aware of others, as well as to provide their own examples.

1.3 Explain coupling and cohesion.

Indicative content

a. Coupling
b. Cohesion
c. How these are implemented in OOP

Guidance

Coupling and cohesion are Software Engineering concepts and can be regarded as aspects of software quality; coupling measures the extent to which one part of the software depends on other parts, while cohesion is a measure of how much parts of the software which are grouped together actually belong together. This part of the syllabus is intended to emphasise that the rationale for OO is to encourage low coupling and high cohesion. Candidates are expected to be able to discuss these two terms and explain how they are operationalised in OOP.

2. Concepts

Learners will be able to:

2.1 Describe techniques for establishing user requirements.

Indicative content

a. Abstraction
b. Encapsulation
c. Data hiding/information hiding
d. Classes and objects (instances)

Guidance

Candidates should be able to compare these concepts and explain how they are different to each other - sometimes the differences are quite subtle. These ideas can be realised in languages which are not object-oriented, so candidates should also be able to explain how they are implemented in OO, as well as their relevance to OOP.

2.2 Explain inheritance and other inter-class relationships.

Indicative content

a. Single, multilevel, multiple, hierarchical and hybrid inheritance
b. Super-classes (base classes)
c. Sub-classes (derived classes)
d. Specialisation vs. generalisation
e. Abstract and concrete classes and methods
f. Inheritance for specialisation vs. specification
g. Inter-class relationships, e.g.:
   i. Isa
   ii. Has-a
   iii. Part-of
   iv. Association
   v. Aggregation
   vi. Composition

Guidance

Candidates may be given a scenario and asked to derive a hierarchy for that scenario. They should understand subtle differences between each term and how they are distinguished from each other. Candidates could also be asked to explain a real-world scenario in which these concepts could be used.

2.3 Describe class members.

Indicative content

a. Fields (data members, variables, attributes) and methods (member functions, procedures)
b. Messages
c. Object state
d. Constructors (parameterised, copy, conversion, default) and destructors
e. Accessors (getters) and mutators (setters)
f. Object and member scope

Guidance

Candidates may be given a scenario and asked to present a class definition which places members in correct sections and gives them the correct visibility. They should be able to understand when it is appropriate to use different kinds of constructor and destructor, as well as the role of accessors and mutators in implementation of information or data hiding or encapsulation. They should also understand advantages and disadvantages of using accessors and mutators to control access to data.

2.4 Explain polymorphism.

Indicative content

a. Ad-hoc and parametric
b. Method overloading
c. Method overriding
d. Operator overloading
e. Templates

Guidance

Candidates may be asked to give examples of ad-hoc or parametric polymorphism. They should understand the relationship between overriding and inheritance, as well as the advantages and disadvantages of ad-hoc and parametric polymorphism.

3. Design

Learners will be able to:

3.1 Use Unified Modelling Language (UML).

Indicative content

a. Use case diagrams, e.g.:
   i. Actors
   ii. System boundaries
   iii. <<uses>>, <<extends>> and <<includes>>

Guidance

Candidates may be given a scenario and asked to draw a Use Case diagram. They will need to understand what the system boundary is and the role of actors and use cases to capture the functionality and requirements of the given scenario. The scenario may also require use of include and extend relationships. Candidates may also be asked to provide a use case description, which will provide a description of the interactions between the actors and use cases, which can also include alternative steps.

3.2 Analyse scenarios using appropriate tools.

Indicative content

a. Class diagrams, e.g.:
   i. Associations
   ii. Aggregation
   iii. Dependency
   iv. Inheritance
b. Object interaction diagrams
c. Object state transition diagrams

Guidance

Candidates may be given a scenario and asked to draw the appropriate diagram depending on whether a static or dynamic view is required. For a Class diagram, they need to be able distinguish what the classes are, showing the attributes, operations and how they relate to each other, plus understand when it is appropriate to apply the concepts of aggregation, dependency and inheritance. For Object interaction diagrams, candidates should be able to show the interactive behaviour of the scenario in the form of either a sequence or collaboration diagram. Key elements are to show the objects which take part in the interaction, messages which flow between them and the sequence in which the messages flow. For state transition diagrams, candidates need to be able to show what states an object can have and what events can cause an object to change state.

3.3 Write and interpret Object Constraint Language (OCL).

Indicative content

a. Invariants
b. Preconditions
c. Postconditions

Guidance

As part of the design of a given scenario, candidates may be required to provide greater precision when defining some of the constraints which are to be applied to the behaviour of some of the objects. Candidates need to be able to construct unambiguous OCL expressions, demonstrating their understanding of concepts of invariants, preconditions and postconditions.

3.4 Explain and analyse design patterns.

Indicative content

a. Pattern documentation, e.g.:
   i. Motivation
   ii. Pre-requisites
   iii. Structure
   iv. Participants
   v. Consequences
b. Examples of patterns, e.g.:
   i. Adapter
   ii. Decorator
   iii. Iterator
   iv. Observer
   v. Singleton
   vi. Factory

Guidance

Design patterns in the context of object-oriented programming are there to provide solutions to common problems found in software design. Candidates need to be aware what information the documentation provides and how they help provide the solution. There are three broad categories of design patterns: creational; behavioural and structural patterns and candidates need to have in-depth knowledge of at least one from each type.

4. Practice

Learners will be able to:

4.1 Describe SOLID.

Indicative content

a. Single-responsibility principle
b. Open-closed principle
c. Liskov substitution principle
d. Interface segregation principle
e. Dependency inversion principle

Guidance

Candidates should be able to explain the implications of each of the five Solid principles. They should be able to demonstrate the implementation of each principle in an object oriented programming language with which they are familiar.

4.2 Construct and understand class hierarchies.

Indicative content

a. Implementing concepts listed in 2.2

Guidance

Candidates are expected to be able to use inheritance and other interclass relationships to construct software artefacts.

4.3 Demonstrate and explain implementation of designs.

Indicative content

a. Implementing designs in an object oriented programming language
b. Refactoring

Guidance

Candidates are expected to be able to take the provided design and show how that design might be implemented in a specific OO language. Designs are usually provided in UML (see 3.1).

4.4 Understand how OO code might be tested in practice.

Indicative content

a. Class testing, constructing class tests from OCL or state transition diagrams, test driver construction.
b. Testing interactions and class hierarchies
c. Black box and white box

Guidance

Candidates may be expected to compare different approaches for testing and to understand their advantages and disadvantages, and where and when it would be appropriate to use them. Candidates may be given a piece of code and asked which may be an appropriate approach to test it, or to provide test cases to show that it is working correctly.

Examination Format

This module is assessed through completion of an invigilated written exam.

Type	Four written questions from a choice of six, each with equal marks
Duration	Two hours
Supervised	Yes
Open Book	No (no materials can be taken into the examination room)
Passmark	10/25 (40%)
Delivery	Paper format only

Adjustments and/or additional time can be requested in line with the BCS reasonable adjustments policy for candidates with a disability or other special considerations.

Question Weighting

Candidates will choose four questions from a choice of six. All questions are equally weighted and worth 25 marks.

Recommended Reading

Primary texts

Title: UML Distilled
Author: M. Fowler
Publisher: Addison-Wesley
Date: 2003
ISBN: 978-0321193681

Title: Growing Object-Oriented Software Guided by Tests
Author: S. Freeman and N. Pryce
Publisher: Addison-Wesley
Date: 2009
ISBN: 978-0321503626

Title: Design Patterns Explained: A New Perspective on Object-oriented Design
Author: A. Shalloway and J. Trott
Publisher: Addison-Wesley
Date: 2004
ISBN: 978-0321247148

Title: The Object-Oriented Thought Process
Author: M. Weisfeld
Publisher: Addison-Wesley Professional
Date: 2013
ISBN: 978-0321861276

Additional texts

Title: Object-Oriented JavaScript
Author: V. Antani and S. Stefanov
Publisher: Pakt Publishing
Date: 2017
ISBN: 978-1785880568

Title: Object-Oriented Programming with C++
Author: E. Balagurusamy
Publisher: McGraw Hill India
Date: 2017
ISBN: 978-9352607990

Title: Objects First with Java: A Practical Introduction Using BlueJ
Author: D. Barnes and M. Kolling
Publisher: Pearson
Date: 2016
ISBN: 978-0134477367

Title: The Unified Modelling Language User Guide
Author: G. Booch, J. Rumbaugh and I. Jacobson
Publisher: Addison-Wesley
Date: 2017
ISBN: 978-0134852157

Title: Beginning C# Object-Oriented Programming
Author: D. Clark
Publisher: Apress
Date: 2013
ISBN: 978-1430249351

Title: Design Patterns: Elements of Reusable Object-Oriented Software
Author: E. Gamma, R. Helm, R. Johnson and J. Vissides
Publisher: Addison-Wesley
Date: 1995
ISBN: 978-0201633610

Title: Object-Oriented Programming with Visual Basic .NET
Author: J. P. Hamilton
Publisher: O'Reilly Media
Date: 2002
ISBN: 978-0596001469

Title: Python 3 Object-oriented Programming
Author: D. Philips
Publisher: Pakt Publishing
Date: 2018
ISBN: 978-1789615852

Using BCS Books

Accredited Training Organisations may include excerpts from BCS books in the course materials. If you wish to use excerpts from the books you will need a license from BCS. To request a license, please contact the Head of Publishing at BCS outlining the material you wish to copy and its intended use.

Document Change History

Any changes made to the syllabus shall be clearly documented with a change history log. This shall include the latest version number, date of the amendment and changes made. The purpose is to identify quickly what changes have been made.

Version Number	Changes Made
Version 1.0 **August 2021**	Document created

CONTACT

For further information please contact:

BCS
The Chartered Institute for IT
3 Newbridge Square
Swindon
SN1 1BY

T +44 (0)1793 417 445

www.bcs.org

The Chartered Institute for IT