Next Generation Higher National Unit Specification

Object-Oriented Programming (SCQF level 8)

Unit code: J7DW 48

SCQF level: 8 (16 SCQF credit points)

Valid from: session 2023–24

Prototype unit specification for use in pilot delivery only (version 1.0) May 2023

This unit specification provides detailed information about the unit to ensure consistent and transparent assessment year on year.

This unit specification is for teachers and lecturers and contains all the mandatory information required to deliver and assess the unit.

This edition: May 2023 (Version 1.0)

Unit purpose

This unit enables learners to develop a broad knowledge of the concepts, principles, and techniques of object-oriented software development. Learners develop problem-solving and object-oriented technical skills, with emphasis on the development and testing of the class libraries required for a given problem domain.

This specialist unit is particularly suitable for learners who are studying software development or computing science. We strongly recommend that learners are familiar with fundamental programming concepts at SCQF level 7. They can demonstrate this by having completed one or more of the following Higher National (HN) units:

- Developing Software: Introduction at SCQF level 7
- Software Development: Developing Small Scale Standalone Applications at SCQF level 7
- Software Development: Programming Foundations at SCQF level 7

On completion of the unit, learners may progress to more advanced topics in object-oriented software design and development.

Unit outcomes

Learners who complete this unit can:

1 investigate object-oriented programming techniques and apply them to a design
2 implement a solution from an object-oriented design using object-oriented techniques
3 test the completed program

Evidence requirements

Learners must provide product evidence. Knowledge is inferred from the product evidence.

Learners must produce evidence of successfully investigating and applying appropriate object-oriented programming techniques.

Learners must implement a given object-oriented design. The object-orientated design must be of sufficient complexity to cover the knowledge and skills for each outcome. The design must consist of at least four classes, and learners must implement at least one ‘one-to-many’ association. They must also show the correct use of encapsulation and inheritance.

Learners must produce completed test documentation that records both the expected results of the test data and the actual results. The test data must test the implemented solution in both scope and range.

Learners must record and evaluate the results of the test runs. Where there are discrepancies between the expected results and the actual results, learners must amend and correct the coding accordingly.

The product evidence can be produced over an extended period of time in lightly controlled conditions. Give learners access to learning materials. Authentication is required when evidence is produced in lightly controlled conditions.

Knowledge and skills

The following table shows the knowledge and skills covered by the unit outcomes:

Knowledge	Skills
Learners should understand: ♦ object-oriented concepts and terms ♦ object-oriented programming techniques ♦ objects and classes ♦ attributes and methods ♦ parameter passing ♦ abstraction, encapsulation and information-hiding ♦ inheritance ♦ polymorphism ♦ association ♦ aggregation and collection ♦ coupling and cohesion ♦ overloading methods	Learners can: ♦ declare and initialise variables ♦ use operators ♦ implement control structures ♦ define data structures ♦ access and manipulate data structures ♦ use parameter passing ♦ create classes ♦ create instances of classes ♦ create relationships between classes ♦ create constructor methods ♦ use exceptions ♦ use standard object libraries ♦ document code ♦ implement a test plan using a defined strategy ♦ maintain test documentation ♦ evaluate results of test runs ♦ amend code as necessary

Meta-skills

Throughout the unit, learners develop meta-skills to enhance their employability in the computing sector.

Self-management

This meta-skill includes:

- focusing: sorting and maintaining documentation throughout development in a logical and efficient manner; attention to detail to ensure error-free, robust program code; considering all aspects of user requirements
- adapting: critically reflecting on personal skills development; self-learning to develop wider skills and extend development beyond requirements and taught content
- initiative: independent thinking to establish user requirements and design solutions; developing an object-oriented application based on client information; self-motivation and taking ownership of personal development; time management

Social intelligence

This meta-skill includes:

- communicating: receiving information to establish the user requirements and ensure an understanding of the brief; giving information during application testing to confirm objectives and ensure that requirements are understood
- feeling: storytelling through the creation of technical documentation, providing a walk-through and detail of the program and its user interface

Innovation

This meta-skill includes:

- creativity: using imagination to provide a solution that is object-oriented; seeing the program solution through the user's eyes; generating ideas and thinking about problem areas and how to provide a solution; visualising to create an overall impression of the completed solution throughout the process
- sense-making: analysis; seeing the bigger picture
- critical thinking: thinking logically to ensure a coherent approach and to meet requirements appropriately

Delivery of unit

This unit is intended to form part of a group award and is not suitable as a stand-alone unit. We have designed the unit as a follow-on to Developing Software: Introduction at SCQF level 7. We assume that learners have knowledge of fundamental programming concepts, but we advise that you briefly refresh these basic topics (variables, operators, iteration, selection, arrays) and show learners how to implement these using the syntax of the object-oriented programming language you have chosen to use.

You should deliver the unit in conjunction with Systems Development: Object-oriented Analysis and Design at SCQF level 8, which develops understanding and skills in the production of object-oriented design documentation. Your centre can also deliver the unit in conjunction with a unit on human computer interaction (HCI), such as Human Computer Interface at SCQF level 8.

We recommend that no more than 6 hours should be allocated to revision topics, leaving the vast majority of the time to be allocated to object-oriented programming techniques. You should cover the following topics:

- defining data structures
- accessing and manipulating data structures
- using parameter passing
- creating classes
- creating instances of classes
- creating relationships between classes
- creating constructor methods
- overloading methods
- use of exceptions
- use of standard object libraries
- use of pre-defined interface components
- documenting code

Additional guidance

The guidance in this section is not mandatory.

Content and context for this unit

This unit is a 2-credit introduction to object-oriented programming. Learners acquire knowledge of the concepts and principles of object-oriented software development. Learners must demonstrate object-oriented programming skills through the creation of object-oriented solutions to problems that require a software solution.

Object-oriented and event-driven programming are not mutually exclusive, and almost all object-oriented languages can be used in an event-driven way. You can structure a program in an object-oriented manner and enable user interaction through an event-driven interface. Learners must therefore have an understanding of basic event-driven techniques for designing programs that have a graphical user interface.

Learners should already be familiar with basic programming building blocks, allowing the unit to focus on object-oriented programming concepts, such as classes and inheritance, to create simple graphical user interface (GUI) applications that match the unit purpose. Your learning and teaching approaches may include problem-based learning, teamwork, and potentially an element of competition such as ‘solve in an hour’ code challenges to engage learners and encourage self-directed study.

We have not specified the implementation language. At the time of writing, the most appropriate languages to use are Java or C#. Both of these languages are widely used in industry and have a similar, C-type syntax. Java is a good teaching language with a large object library, and the creation of user interfaces is supported through the Abstract Window Toolkit (AWT) and Swing.

C# allows for the creation of interfaces through simple drag and drop, and the Microsoft XNA development kit can be used in association with C# for the development of fairly complex computer games, if you want to enable learners to develop games. You should introduce learners to the standard object libraries and show them Sun Java Docs for Java, or Microsoft Developer Network (MSDN) libraries for C# and C++, and demonstrate how to make use of appropriate standard objects through library calls such as ‘import’ or ‘using’.

We recommend that you introduce learners to technical skills related to event-driven programming, depending on which implementation language you have chosen. For example, if you use Java then you can introduce AWT and the Swing component classes. If you use C#, you can show learners how to set up user interfaces through code or use the toolbox of user interface components and their various properties. The GUI libraries supplied with modern object-oriented languages are an excellent way of introducing the idea of classes and objects. The hierarchies can aid in the understanding of inheritance and encapsulation. In addition, you can use the interactive event methods to introduce the concept of events in software development.

To understand practical object-oriented programming techniques and structures, there are several theoretical object-oriented concepts that learners should understand. You must cover the following topics:

- objects and classes
- attributes and methods
- parameter passing
- abstraction, encapsulation and information-hiding
- inheritance
- polymorphism
- association
- aggregation
- coupling
- cohesion

We recommend that you combine theory and practice as much as possible, rather than separating them. For each of the above topics, you should provide learners with several practical exercises. You should increase the complexity of the practical exercises as learners progress and learn new techniques. Providing examples that show a new technique or skill in isolation is useful for initial understanding of how a technique may be implemented, but is not useful for the problem-solving and program design process. You should adopt an approach that incorporates new techniques into larger programs from an early stage in the unit so that there is opportunity for integration of concepts.

Internal documentation of code must adhere to current commercial standards. At the time of writing, this includes basic Javadoc for Java, or basic XML documentation for C#. You should stress the importance of naming conventions, indentation, and version control when discussing code structure and documentation.

You should help learners to develop the ability to analyse a project design brief and implement a suitable object-oriented solution through practice, rather than teaching this as an explicit skill. In outcome 2, you should introduce the object-oriented software development methodology and unified modelling language, and we recommend that you also introduce learners to case-based reasoning. This is the technique of looking back at previous problems and their solutions and using similar cases as the basis for the design of an implementation for the new problem. For example, you could show learners the problem of writing a noughts and crosses game, and as a class they could design and implement the solution to this problem. You could then ask learners to create a Connect 4 game, using the noughts and crosses game as a basis for their design, due to the many similarities between the two games.

As a non-introductory programming unit, the context and examples you use should be relatively complex in terms of the scope and programming fundamentals required. However, they should be relatively simple in terms of object orientation. Using examples of relatively simple games (for example noughts and crosses, Battleship, Pac-man, Pong, Space Invaders) would hopefully make the learning process more enjoyable and meaningful for

learners and would permit them to concentrate more on the programming techniques involved, rather than trying to understand the initial problem.

The GUI libraries supplied with modern object-oriented programming languages are an excellent way of introducing the idea of classes and objects. If you adopt this approach, then you might also introduce learners to HCI design techniques, such as storyboarding.

By the end of the unit, learners should have achieved a good foundation in the skills required for developing reliable, robust and efficient object-oriented program designs for simple applications.

The unit covers some of the skills described for a pre-entry or junior technician role in the National Occupational Standards IT and Telecoms (2009). The main areas covered correspond to:

- Discipline 5.2: Software Development
- Discipline 5.3: IT/Technology Solution Testing

There are also opportunities in the unit to address a range of skills at both foundation and intermediate level that are described in the National Occupational Standards for IT Users v3. The most likely areas to be covered would be ‘Using the Internet’ and ‘IT Software Fundamentals’.

Approaches to assessment

We recommend that you assess the unit with a single project covering all three outcomes. Ideally, the assessment project would involve producing an application or a library of routines. You can also offer two or more projects to give learners a degree of choice. The evidence requirements can be met through one large program that covers all knowledge and skills, or via a portfolio of two or more smaller programs.

Equality and inclusion

This unit is designed to be as fair and as accessible as possible with no unnecessary barriers to learning or assessment.

You should take into account the needs of individual learners when planning learning experiences, selecting assessment methods or considering alternative evidence.

Guidance on assessment arrangements for disabled learners and/or those with additional support needs is available on the assessment arrangements web page: www.sqa.org.uk/assessmentarrangements.

Information for learners

Object-Oriented Programming (SCQF level 8)

This information explains:

- what the unit is about
- what you should know or be able to do before you start
- what you need to do during the unit
- opportunities for further learning and employment

Unit information

This non-introductory unit covers the object-oriented programming skills required for a career in software development. We assume you have prior knowledge and proficiency in basic programming concepts and techniques. We recommend that you have successfully completed Developing Software: Introduction at SCQF level 7 before starting the unit.

In the unit you acquire knowledge of the concepts, principles, and techniques of object-oriented software development that are necessary for you to design and develop object-oriented software. This involves the following areas of learning:

- Investigate object-oriented programming techniques and apply them to a design.
- Implement a solution from an object-oriented design using object-oriented techniques.
- Test the completed product.

Your knowledge and skills are assessed by the following tasks and associated learning:

- Using the features of an object-oriented programming language, you implement a software solution based on a given design.
- Your understanding and grasp of object-oriented concepts and programming techniques are reinforced throughout with practical exercises.
- Using a test plan, you test your software to ensure it works correctly and meets the user requirements.
- You amend any errors in your code to achieve a robust, reliable and efficient working program.

The theory and practice of programming for an object-oriented design are intertwined in your learning experience. The applications that you develop are simple, but require the implementation of object-oriented programming techniques and processes.

Throughout the unit, you develop meta-skills in the areas of self-management, social intelligence and innovation.

When you finish the unit, you may progress to more advanced units in object-oriented design and programming at SCQF level 8 and above.

Administrative information

Published: May 2023 (version 1.0)

Superclass: CB

History of changes

Version	Description of change	Date

Note: please check SQA's website to ensure you are using the most up-to-date version of this document.