M250 Object-oriented Java Programming

Combined glossary: Units 1–12

Glossary

The definitions in this glossary are written assuming that you have completed all twelve units of M250.

* denotes glossary entry presented here has been amended from that presented in the main bindings.

absolute pathname
A full path to a file beginning from the name of the storage device.

abstract class
A class that defines a common message protocol and common set of instance variables for its subclasses. In Java an `abstract` class cannot be instantiated.

abstract method
A method declared as `abstract`. Abstract methods have no bodies. They must be implemented in any concrete subclasses of the class in which they are specified. Interfaces also specify abstract methods, which must be implemented by classes implementing the interface.

access level
The scope for access that exists for a software component such as a method. There are four access levels in Java: `public`, `private`, `protected` and default. The default level occurs when no access modifier is specified.

access modifier
One of three Java keywords (`public`, `private`, `protected`) which specify the visibility of variables and methods.

accessibility
See *visibility*.

accessor message
Another name for a *getter message*.

accessor method
Another name for a *getter method*.

actual argument
An actual argument is a value used in a message that is copied to a formal argument for use inside the corresponding method. Actual arguments must be of a compatible type with their corresponding formal arguments.

annotation
An annotation is a piece of code in a source file that indicates a programmer's intention to the compiler. The only annotation we use in M250 is `@Override`. All annotations begin with the character `@` in Java.

application domain
See *problem domain*.

argument
See *actual argument* and *formal argument*.

array
An indexable, fixed-size collection whose elements are all of the same type.

array initialiser
An expression that can be used to initialise an array using assignment, a shortcut way of initialising array elements.

`assert`
The keyword in a Java assertion statement used to indicate a condition that should be `true` in order for the code to be correct, particularly used in `private` methods.

assertion
A statement in the Java language that enables you to test your assumptions about your program; a condition that a programmer believes should be `true`.

assign
See *assignment*.

assignment
Assignment is the process that results in the variable on the left-hand side of the *assignment operator* receiving a copy of the value on the right-hand side of the assignment operator. The value may be a primitive value or a reference to an object. If the right-hand side of the assignment operator is an expression, it is evaluated first.

assignment operator
An operator (=) used to assign a value to a variable in an assignment statement.

assignment statement
A statement that assigns a particular value to a variable (see *assignment*).

attribute
Some property or characteristic of an object that can be accessed using a getter method. Attributes are generally implemented by instance variables. Examples of such attributes are `position` and `colour` for `Frog` objects, and `balance` and `holder` for `Account` objects.

attribute value
The current value of an attribute, often the same as an instance variable value, but possibly computed using instance variables.

auto-boxing
This is the automatic process of wrapping primitive values when called for by the context of usage in a Java program.

auto-unboxing
This is the automatic process of unwrapping primitive values when called for by the context of usage in a Java program.

automatic type conversion
Where the Java compiler converts a value of some type to another type without the need for any explicit conversion of type on the part of the programmer. Automatic type conversion occurs in certain contexts, such as in an assignment statement when a compatible type on the right-hand side of the

assignment is converted to the type of the variable on the left-hand side of the assignment.

behaviour
A term used to describe the way in which an object behaves in response to the messages in its protocol.

binary digit
Either of the two digits 0 and 1 in the binary number system. Binary digits are used for the internal representation of numbers, characters and instructions. The binary digit is the smallest unit of storage.

binary operator
An operator that has two operands.

bit
An abbreviation of *binary digit*.

block
See *statement block*.

body
Another word for a statement block; for example, the body of a `while` loop is a statement block.

Boolean condition
A Boolean expression used to control the conditional execution of a statement block.

Boolean expression
An expression that evaluates to either `true` or `false`.

Boolean operator
An operator used to combine simple Boolean expressions to form more complex Boolean expressions, which in turn can be combined with other Boolean expressions. They are also known as *logical operators*.

bucket
A data structure that can contain zero or more unsorted elements. A `HashSet` uses buckets to implement a set. The hash code of each incoming element is used to decide which bucket to store it in.

buffer
Could refer to either an area used for temporary storage as data is transferred between a data source and a data sink, or a sequence of unfilled components that allows a `StringBuilder` object to grow.

bug
The cause of a run-time error.

bulk operations
Operations that act on an entire collection without requiring the programmer to focus on individual elements.

bytecode
Bytecode is the intermediate code produced by the Java compiler. In BlueJ, compilation is done when the `Compile` button is pressed. This will create a

bytecode file, for example, `Frog.class`, from the source code file `Frog.java`. The bytecode file is portable because each computer that can run Java programs has a Java Virtual Machine that understands bytecode and converts it into the machine code required for that particular computer. Bytecode is so-called because its instructions are a byte in size.

call stack
A representation of the order of methods called at some point during the execution of a program, shown as a stack of method names, with the first called method on the bottom and the last called method on the top.

capacity
The number of characters a `StringBuilder` object can hold.

casting
The prefixing of a value with the name of a type in parentheses in order to convert a copy of that value into a different type. For example, `(int) 99.0` casts the value `99.0` into an `int` value.

catch
The process of catching an exception, and the keyword introducing the clause in a *try-catch* statement that handles an exception.

chaining
Constructors are said to be chained, meaning that when an object is created, the constructors of all its superclasses are also called, either explicitly or implicitly.

checked exception
An exception that the compiler requires the programmer to explicitly handle or declare may be thrown in a method header using a `throws` clause. These exceptions are ones that a programmer may reasonably be expected to catch or should be made aware of.

class
A class is like a blueprint for the creation of objects and ensures that all its instances have the same instance variables and behave in response to messages in an identical manner.

class header
The line in a class definition which gives its access modifier, name and, optionally, the name of a class it extends and the name(s) of any interface(s) it implements. Example usage:

```
public class WeatherFrog extends Frog implements
WeatherClient
```

class method
A class method is a method declared with the keyword `static` and is associated with a class rather than with any of its instances. Classes are not objects so code in a class method cannot use the expression `this` or the keyword `super`. Class methods are invoked directly on the name of the class, using static binding, and so do not exhibit polymorphism.

class variable
A class variable is a variable declared with the keyword `static`. A class variable is associated with a class rather than with any of its instances, although each instance of a class can access its own class's class variables. A class only ever has one copy of each of its class variables. See *qualified* for details of how class variables are accessed.

client (class)
In programming, an object that uses a service provided by some other (server) class.

collaboration
The achievement of a software solution using two or more communicating objects.

collection *
A collection consists of a number of, possibly zero, objects (strictly object references) or primitives. The objects or primitives within a collection are referred to as elements.

Collection Classes/collection classes
A set of library classes used to create various kinds of collection such as tree structures and queues. When capitalised, this phrase usually refers to classes that are part of Java's Collections Framework, which excludes array types. When used in lower case, the same phrase generally means any class that implements a collection (in the looser English language sense), which includes array types.

Collections Framework
The Collections Framework includes a set of collection interfaces (`Collection`, `Set`, `List`, `Map`, and others), a set of collection classes (e.g. `HashSet` and `HashMap`) that implement them, and some supporting utility classes (e.g. `Collections`). The framework also includes a set of associated recommendations about constructors and about expected behaviours in common. Arrays are *not* part of the Collections Framework.

comma-delimited file
A file whose items of data (tokens) are separated by commas.

comment
A comment is a piece of text in program code to assist human readers in understanding the code, and which the compiler ignores. In Java multi-line comments are delimited by `/*` and `*/`. End-of-line comments begin with two forward slashes (`//`) and continue to the end of the line on which they begin. Javadoc comments are placed between the symbols `/**` and `*/` and must appear immediately before a method, constructor or class header.

common message protocol
A set of messages shared by a number of classes. Often used to describe the set of messages specified by an abstract class for its concrete subclasses.

`Comparable`
An interface that allows an element type to be sorted. It contains a single method: `compareTo()`.

`compareTo()`
The sole method of the `Comparable` interface. Allows classes of objects implementing this interface to be sorted (i.e. gives them a natural ordering).

comparison operators
A set of operators used for comparing values of expressions, including ==, > and < .

compilation error
An error detected by a compiler when code is compiled (which is known as compile-time). No bytecode is generated if there are compilation errors.

compile-time error
See *compilation error*.

compiler
Software which checks that text written in a high-level language is correctly formed and, as far as can be determined before compilation, that it is meaningful source code for the language.

component
See *software component*.

component (of an array)
The memory location at which an element (or a reference to it) of an array is stored.

component type
The type of the components of an array object; this determines the types of the object or primitive value that can be stored in the array.

compound expression
An expression built up using other sub-expressions, for example, the following is a compound expression: `(3 + 2) * (6 - 3)`.

concatenation
The joining of two strings. In Java the string concatenation operator is + (the plus sign). For example, `"Milton " + "Keynes"` evaluates to `"Milton Keynes"`.

concrete class
A class which is not abstract; a class for which instances can be created.

condition
See *Boolean condition*.

conditional selection
The use of `if` statements to select and execute alternative statement blocks based upon the value of a Boolean condition.

constant
A constant is a variable whose value is fixed and unchangeable. Normally the keyword `final` is used to make a value into a constant. In addition, where only a single value is needed for a class, constants are typically declared as `static`. However, `static` is not always used for constants, as sometimes a

situation needs to be modelled where each instance of a class has its own different constant value, such as a serial number.

constant instance variable
Constants are usually declared as `final static` variables. However, sometimes it makes more sense to define a constant as a `final` instance variable. See *constant* for more information.

constructor
A programming construct, similar to a method, used to initialise a newly created object.

constructor chaining
The process whereby constructors use `super()` to invoke constructors higher up their inheritance hierarchy.

convention
A commonly followed rule for implementing some feature of a software system that is not enforced by the language used, compiler, or platform on which the software is used.

data field
A term encompassing instance variables and class variables as well as constants.

data hiding *
The use of access modifiers to restrict access to class members, particularly instance variables, so that an object becomes a ‘black box’, with access to the encapsulated data (the instance variables) being possible only through a limited set of public methods.

DbC
See *design by contract*.

debugging
The identification and removal of errors (bugs) from a program.

declaration
A statement in which memory is reserved for a variable of some type that is formed by writing the type of the variable followed by some name (its identifier).

default constructor
A zero-argument constructor that simply invokes `super()`. This constructor is provided automatically by the compiler when the programmer has not specified any constructor in a class.

default values
The values used by default to initialise instance variables of a class, which may be overwritten by another, explicit, initialisation.

defensive programming
A programming technique in which a method provides alternative paths to deal with expected and unexpected arguments. The method may return a Boolean to indicate success or failure, or (when necessary) use an exception to signal a failure.

delimiter
A character used to mark where some part of a program starts or ends. For example, in Java, the character `;` marks the end of a statement and the character `{` marks the beginning of a statement block.

design by contract (DbC)
A programming technique in which a specification states a precondition for the correct use of a method, and a postcondition that the method will achieve in the event that its precondition is met. An exception is thrown by a method when its precondition is not met.

design principle
In contrast to a guideline or suggestion, the word principle is reserved for recommendations that apply universally or nearly universally. In this module, the term design principle is applied to principles governing how code should be organised. The authors reserve the right sometimes to violate design principles for teaching reasons!

destructive operation *
An operation on a collection that changes the contents of the collection in some way is said to be destructive. You may need to examine the Java API to determine whether a method is destructive or non-destructive.

dialogue box
A type of window with buttons through which users can be given information by a program or provide information to a program on request. In M250, dialogue boxes are provided by class methods of the `OUDialog` class.

diamond operator
From Java 7 onwards the diamond operator, written <> , simplifies the construction of collection objects. The compiler uses type inference to determine the appropriate type of the collection based on the type of the collection's reference variable.

difference
For sets, the difference of *A* and *B* is all the elements in *A* that are not in *B*. This result is usually different from the difference of *B* and *A*. The term is used to refer both to the result and to the operation.

direct access (of an instance variable)
Accessing an instance variable using its name, rather than using a getter method.

direct interaction
Direct interaction is not a formal technical term, but a descriptive phrase used to describe a situation where one object has a reference to another, and this reference is used to affect the state or behaviour of the other object. Contrast with *indirect interaction*.

direct subclass
A class is a direct subclass of another class if it is directly below that class in the class hierarchy.

direct superclass
A class is a direct superclass of another class if it is directly above that class in the class hierarchy. In Java the keyword `extends` is used by a subclass to indicate its direct superclass.

domain
See *problem domain*.

dynamic binding
The postponing of what method to select for execution in response to a message until run-time. The method selected depends on the class of object that receives the corresponding message, rather than on the type of the variable that references it.

dynamic compilation
A compilation technique (used by the Java environment) generating real machine code from bytecode (intermediate code). A chunk of bytecode is compiled into machine code just prior to being executed. (This is different from, and faster than, the piecemeal operation of an interpreter.) The real machine code is retained so that subsequent execution of that chunk of bytecode does not require the translation to be repeated.

effective length
The number of meaningful elements that a fixed-size collection actually holds.

element
The name given to the primitive value stored in, or the object referenced by, an array component.

encapsulation
Encapsulation is the parcelling up of information and behaviour into a reusable component. Objects allow you to *encapsulate* data by incorporating into a single entity (the object) both the data (instance variables) and the behaviour (methods) defined for that data.

end-of-line comment
See *comment*.

entry
An entry in a map means a key–value pair; that is, it refers to the combination of a key and its associated value.

`equals()`
A method that determines whether two objects are equal or not. Since it is implemented by the class `Object`, all classes of object understand the message `equals()`, but the method is overridden to define versions of equality appropriate to particular classes. For many classes, such as `String`, the method `equals()` is defined effectively to mean ‘has the same state as’. More generally, `equals()` is coded to mean ‘should be regarded as equal for our purposes’.

escape character
A character used to mark the start of an escape sequence in a string. In Java this is the `\` (backslash) character.

escape sequence
A sequence of characters beginning with a special escape character, such as a backslash, that allows subsequent characters to take on a different meaning for the compiler.

evaluate
To find the value of something. We say that expressions have a value or evaluate to some value.

exception
An object that is thrown by a method or the JVM as the result of a run-time error. The exception object holds details of what went wrong, allowing the exception handler that catches the exception to take appropriate action, or provide a useful error message to the programmer.

exception handler
A block of code written to deal with (handle) an error that has been signalled within some program code. In Java, the error must have arisen in the context of a `try` block, and the exception handler is in an associated `catch` block. The `try` and `catch` together make up a single statement.

exception handling
The programmed catching of an exception and the subsequent execution of code to abandon or continue execution, or to restore the software to a meaningful state.

expression
Code that evaluates to a single value. Expressions can be formed from literals, variables, operators and messages.

failing fast
The programming philosophy which recommends that errors be signalled as soon as possible so that their cause can be more easily traced, and so that errors are not allowed to be passed on to other parts of a software system, potentially causing problems later. In Java this is done by throwing an exception.

field
See *data field*.

`File`
An instance of this class contains the pathname to a file or folder in a system-independent format. The file specified need not exist; instead the pathname may point to a *potential* new file or folder.

`final`
The keyword `final` prevents a variable from ever having its value reassigned once it has an initial value.

`finally`
Java keyword that introduces a block of code in a *try-catch* statement that will always be executed, regardless of whether an exception occurs or not.

fixed-size collection
A collection whose number of elements is fixed on creation. Such a collection can neither grow nor shrink.

floating-point type
A type capable of representing a decimal number (to a restricted level of accuracy).

flushing
The process of emptying a buffer so that no data remains in it, the data being transferred to a sink.

`for` **statement**
A statement allowing a statement block to be repeatedly executed according to the value of a loop control variable (typically of type `int`) and a Boolean condition whose value is dependent on the loop control variable. `for` loops are typically used when a fixed number of iterations is required. See also *`while` loop*.

for-each **statement**
A statement that enables iteration through every element of a collection.

formal argument
A typed identifier used in a method signature between parentheses to stand for a value (an actual argument) that is passed into the method body by a message.

formatting guidelines
A set of guidelines which specify how program code should be laid out.

garbage collection
The process of destroying objects that have become unreachable because they are no longer referenced by variables, in order to reclaim their space in memory. In certain programming languages, including Java, this process is automatic.

generics
A programming language feature that allows the writing of code that supports many types, but allows a particular type to be specified, so allowing for more type checking to take place. Java collections from Java 5 onwards support generics. When creating such a collection an element type is specified.

getter message
A message that returns as its message answer the value of one of the receiver's attributes. See also *setter message*.

getter method
A method whose purpose is to return the value of one of an object's attributes as its message answer. For example, the method `getPosition()` is a getter method that returns the value of the instance variable `position` held by instances of the `Frog` class.

graphical representation
A visible representation of a software object that, by definition, is invisible.

handler
See *exception handler*.

hash code
In Java, a hash code is an `int` that can be calculated for any object, by sending it the message `hashCode()`. This method is inherited from `Object`, but in general must be overridden if `equals()` has been overridden. Hash codes are used by sets and related collections to allow them to store elements efficiently and to check rapidly for duplicates. If a class redefines `equals()`, then the `hashCode()` method for that class must take the definition into account; otherwise the type will not behave properly in collections.

helper method
A method whose purpose is to do some internal work of an object and which therefore should have `private` access so that it cannot be used by objects of other classes.

hiding
The situation in which a superclass member is not directly accessible in a subclass due to the subclass defining a member with the same name (for a variable) or same signature (for a method).

high-level language
A language (for example, Java) whose structure reflects the requirements of the problem, rather than the facilities actually provided by the hardware. It enables a software solution to a problem, or a simulation of an aspect of reality, to be expressed in a hardware-independent manner.

homogeneous collection
A collection in which all the elements are of the same type. All collections in Java are homogeneous.

identifier
The name given to a *variable*, *method* or *formal argument*.

identity
Identity refers to an object rather than its state. Two reference variables have the same identity if they both reference the same object; that is, if they both contain a copy of the same reference value.

`if` **statement**
A programming construct whereby one or more statement blocks are conditionally executed, according to whether a Boolean condition evaluates to `true` or `false`.

immutable
An object whose state cannot be changed. An object whose state can be changed is said to be mutable.

implement
In software development, to write the program code for some task or specification. We also say that a class implements an interface when it `implements` the interface and is able to understand all the messages specified by the method signatures of the interface.

implementation class *
A class that implements an interface, also known as an implementing class. For example, in the context of collections, `ArrayList` is an implementation class for the `List` interface, it is an implementing class of `List`.

implementing class *
See *implementation class*.

`implements`
A keyword in Java used in a class header to specify that the class implements a particular interface. Example usage:
`public class SomeClass implements SomeInterface`

index
An integer that is used to access the elements of an indexable collection.

indexable collection
A collection in which every element is accessed by an index.

indirect interaction
Indirect interaction is not a formal technical term, but a descriptive phrase used to describe a situation where one object affects the state or behaviour of the other object, but without actually having a reference to that object – i.e. some intermediary object is used. Contrast with *direct interaction*.

indirect subclass
A class is an indirect subclass of another class if it inherits from that class via one or more intermediate classes.

indirect superclass
A class is an indirect superclass of another class, if it is above that class in the class hierarchy but not directly above it.

inheritance *
A relationship between classes by which they are organised into a hierarchy. According to the Java Language Specification (JLS), classes lower in a hierarchy are said to inherit all the *accessible* members from classes higher in the hierarchy, *other than those they override or hide*. However, in general object-oriented terms, an object of class `T` is said to inherit *all* the members of the superclasses of `T`.

initialisation
See *object initialisation* and *variable initialisation*.

`InputStream`
The base class of a hierarchy of classes including `FileInputStream`, `BufferedInputStream` and `ObjectInputStream`, which are used to read (8-bit) byte streams. `InputStream` classes are used to read binary data.

inspector
An inspector is a tool used in M250 to look at the value held (or object referenced) by a variable declared in the OUWorkspace. In the case of a variable referencing an object an inspector will show the internal state of that object, listing its attributes and their current values.

instance
An object that belongs to a given class is described as an instance of that class.

instance method
Code that is executed as the result of a message being sent to an object.

instance variable
A variable whose identifier and type is common to all the instances of a class, but whose value is specific to each instance. Each instance variable either contains a reference to an object or contains a value of some primitive type. For example, `Frog` objects have the instance variables `colour` and `position`. The values of the instance variables of a particular object represent the state of that object.

integrated development environment (IDE)
A software tool that supports the development, compilation and execution of computer programs. BlueJ is an example of an IDE that supports the development of programs in Java.

`interface`
(Java specific) An `interface` specifies a list of messages that a group of unrelated classes, which are said to implement the interface, must be able to receive. An interface lists only method headers (no method code), cannot declare any instance variables and cannot be instantiated. Classes implementing an interface must be able to receive all the messages corresponding to methods specified by that interface, either by defining the required methods or inheriting them.

intermediate code
See *bytecode*.

interpreter
A program that translates statements (rather than translating a whole program) from a high-level language, such as Java, to a machine code language, and executes the machine code.

intersection
Intersection is a mathematical operation on two or more sets that results in a set containing only the elements common to both sets. The term is used to refer both to the result and to the operation.

invocation
Executing code in a method. Class (`static`) methods are invoked directly on a class, whereas instance methods are invoked by sending messages to objects. Also called *method invocation*.

iteration
Also referred to as repetition. The repeated execution of a statement block for as long as some Boolean condition evaluates to `true`. Examples of programming constructs in Java that support iteration are `while` and `for` loops.

Java edition
A category of Java targeted at a particular platform or domain, such as desktop systems or mobile devices.

Java Language Specification (JLS)
The document that specifies the terminology used by the Java language and describes the manner in which various parts of the language work.

Java Runtime Environment (JRE)
The Java Runtime Environment (JRE) consists of the virtual machine (JVM) plus additional software, such as class libraries, that are needed to support the running of Java programs on different platforms.

Java version
A release of an edition of Java.

`java.util.Arrays`
A Java utility class that contains `static` methods for manipulating arrays.

Javadoc
A programming tool that comes with Java. The Javadoc tool picks up information from specially formatted comments and other parts of the class code such as the constructor and the method headers. These are all used to create an HTML file, which describes the class in a standard way. This description is aimed not at the Java compiler, but at human readers.

Javadoc comment
See *comment*.

JLS
See *Java Language Specification*.

just-in-time compilation
See *dynamic compilation*.

key
A key is a value that can be used to uniquely identify any object from a given map-based collection. What constitutes a good key may depend on the purpose of the collection.

key set
The set of all the key values for all the entries in some map. A key set is typically used to iterate over a map-based collection.

key–value pair
Maps can be viewed as a set of key–value pairs. Each entry in a map is a key–value pair. For example, a key might be a person’s name, and its associated value their telephone number.

keyword
A Java keyword is a word reserved for use in the language. Keywords cannot be used as variable identifiers.

lazy evaluation
A type of evaluation used in compound Boolean expressions involving the `&&` and `||` operators whereby the second operand is evaluated only when it is

necessary to do so in order to determine the truth value of the whole expression.

length
The number of elements an array can hold.

list
An ordered collection of variable size that permits duplicate elements.

literal
A textual representation of a primitive value or object. For example, `'X'` is a `char` literal, `4.237` is a double literal and `"hello there!"` is a `String` literal.

local variables
A variable that is declared inside a statement block, such as inside a method body, or `for` statement, or, in the case of M250, the OUWorkspace. The scope of such a variable is restricted to the statement block in which it is declared.

logical error
An error in logic, such as iterating for an incorrect number of times, or indexing an array at an incorrect position. Such an error may, or may not, lead to a program crash. Such errors may be detected while writing source code or during the running of a program, or not at all.

logical operator
An operator whose operands are Boolean values and which evaluates to a Boolean value; for example, the operator `&&`.

loop
In programming, a loop is a sequence of statements that is continually repeated until some Boolean condition is reached.

loop control variable
A variable used to regulate the number of times a loop such as a `for` statement or `while` statement is executed; such a variable is typically declared inside a `for` loop's header or before a `while` loop.

low-level language
A language written for direct programming of a computer's hardware. Each type of computer hardware needs its own low-level language.

maintainability
This is the extent to which a program can easily be maintained over a period of time during which the requirements for the software may change.

map
A collection type that relates a set of keys to their corresponding values.

mapping
Another name for a key–value pair.

member
The term member is a convenient word sometimes used to cover all of the four following categories: class variables, class methods, instance variables

and instance methods. Constructors are *not* members. Members are involved in inheritance.

message
A message is a request for an object to do something. The only way to make an object do something is to send it a message. For example, the position of a `Frog` object changes when it is sent the message `left()` or `right()`; to obtain information on the value of a `Frog` object's `colour` attribute, you send it the message `getColour()`.

message answer
When a message is sent to an object, then, depending on what the message is, a message answer may be returned. A message answer is a value or an object; it is not a message. Sometimes a message answer is used, sometimes it is ignored. A message answer may be used subsequently as the receiver or argument of another message. Getter messages return the value of an attribute, as with the message `getColour()`, which returns a value such as `OUColour.GREEN`.

message expression
A message-send that evaluates to some value, i.e. the message returns a message answer. We also say that it returns a value.

message name
The name of a message, including the following parentheses, but excluding any arguments. For example, the name of the message `left()` is `left()`, and the name of the message `upBy(6)` is `upBy()`.

message-send
The code that sends a message to an object – for example, `frog1.right()`, which consists of the receiver followed by a full stop and then the message.

method
The code that is invoked by the Java Virtual Machine at run-time when an object receives a message.

method body
That part of a method enclosed by braces that follows the method header. The body of a method is an example of a statement block.

method header
A method header consists of an access modifier (e.g. `public`), a return value (e.g. `int` or `void`) and a name (e.g. `setPosition`) followed by any formal argument names enclosed in parentheses (e.g. `(int aNumber)`). For example, the method header for the method of the class `Frog` whose name is `setPosition()` is `public void setPosition(int aNumber)`.

method invocation
At run-time, selecting and executing a method when an object receives a message. See *invocation*.

method signature
The name of the method together with the parentheses and the types of any arguments. For example, the signature for the `setPosition()` method in the `Frog` class is `setPosition(int)`.

microworld
A computer-based simulation with opportunities for manipulation of content and practice of skills.

modal (dialogue box)
A dialogue box that will not allow you to interact with another part of the program until you have responded to it by clicking one of the buttons presented. All the dialogue boxes provided by the class `OUDialog` are modal dialogue boxes.

model (verb)
To simulate an entity in the problem domain using software.

multi-line comment
See *comment*.

mutator message
Another name for a *setter message*.

mutator method
Another name for a *setter method*.

natural ordering
In Java, an element type is said to have a natural ordering if it implements the interface `Comparable` and consequently has a `compareTo()` method. For numbers and strings, the natural ordering corresponds simply to everyday numerical order and alphabetical order.

`new`
A keyword used to create an object – used in conjunction with a constructor.

non-destructive operation
An operation on a collection that does not change the contents in any way.

object
An instance of a class. An object has both state, represented by its attributes, and behaviour, its responses to the set of messages it understands (its protocol). An object models a part of a solution to a software problem.

`Object`
The top-level class in Java. Every class in Java other than `Object` is either a direct or an indirect subclass of `Object`.

object diagram
An object diagram represents an object. It shows the class of the object, its state in terms of attribute values, and its protocol.

object initialisation
The state of an object when it is first created depends on its initialisation, a process whereby its instance variables are set to known values. This initialisation is usually carried out by the constructor in the object's class.

object technology
A synonym for *object-oriented technology.*

object-oriented programming.
An approach to programming in which a software problem is modelled using classes of objects that can achieve a solution by sending messages to one another.

object-oriented technology
The technology associated with viewing software as being made up of objects.

operand
An expression provided to an operator. A binary operator such as + has a left-hand operand and a right-hand operand.

operating system (OS)
The software that manages the resources of a computer, including controlling the input and output, allocating system resources, managing storage space, maintaining security and detecting equipment failure.

operator
A symbol used as a function, that is, it has one or more values it operates on and returns a value.

orchestrating instance
An orchestrating instance is a separate object used to tie together the different parts of a complicated interaction that do not seem to belong to any single one of the objects involved.

ordered collection
A collection where each element in the collection has a well-defined place, as indicated by its index number. Thus, an ordered collection allows us to access and find the first, second or last element etc. However, the order does not have to reflect anything to do with the elements themselves. It is simply determined by where each element has been placed in the list. This contrasts with a sorted collection.

`OutputStream`
The base class of a hierarchy of classes including `FileOutputStream`, `BufferedOutputStream` and `ObjectOutputStream`, which are used to write (8-bit) byte streams. `OutputStream` classes are used to write binary data.

overloading
A method is said to be overloaded when there are other methods, defined in the same class, or inherited (in the JLS sense) from some superclass, with the same name but a different method signature, i.e. different types and/or numbers of arguments.

`@Override`
A Java annotation used to indicate that a method is intended to override an inherited method.

overriding
The process of redefining (replacing) a method that would have been inherited (JLS definition) from a superclass so as to cause it to have different behaviour. A method that overrides a method from a superclass has the same signature and return type as the superclass method.

parameter
A synonym for *argument*.

pathname
A textual representation of how to find a file or folder on a particular operating system.

peripheral device
Any part of the computer that is not part of the essential computer (i.e. the CPU and main memory), such as a mouse, keyboard or flash drive.

persistence
The ability of objects or other data to continue in existence after a program has stopped executing.

persistent
Stored in a non-volatile form, that is, a form that is not lost when the computer's power is turned off. The most common form of persistent storage is a file on a disk drive.

polymorphic method *
A method with the same *signature* as some other method, but which typically defines different *behaviour*. For example the method `home()` defines different behaviour for `Frog`, `Toad` and `HoverFrog` objects. In Java, polymorphism occurs when classes in inheritance hierarchies override methods as well as when several classes implement some interface.

postcondition
A condition that is true after some code (particularly a method) is executed.

postfix operator
An operator that appears after its operand(s).

precedence rules
Rules for determining the order in which various operators are evaluated in a complex expression.

precondition
A condition that must be true before some code is executed in order for that code to behave correctly. The user of a method typically checks its precondition before using it.

primary sort
An initial sorting according to a particular key.

primitive data type
A set of values together with operations that can be performed on them. The primitive data types in Java provide a set of basic building blocks from which all of the more complex types of data can be built. Java’s primitive data types include `int`, `char` and `boolean`.

primitive type variable
A variable declared to hold a value of the declared or compatible primitive data type. Less formally we refer to a primitive variable.

primitive value
A value of some primitive type; for example, `true`, `-1.3`, `42`, or `'x'` are all primitive values of different primitive types.

`private`
An access modifier that restricts access to a class member to objects of the class to which it belongs.

private protocol
The part of the protocol of a class or object that is inaccessible from a context outside of the class itself.

problem domain
The collection of real-world entities within the application area that exhibit the behaviours that the required system has to model.

procedural programming
An approach to programming in which a problem is broken down into smaller, simpler procedures, often incorporating global data structures.

program
A software solution to a problem, typically on a small scale (compared to larger application software).

`protected`
Java access modifier used to restrict access of a member of a class `X` to subclasses of `X` and classes within the same package as class `X`.

protocol
The set of messages an object understands.

`public`
An access modifier applied to a class member that allows all classes of objects to have access.

public protocol
The part of the protocol of a class or object that is accessible from outside of the class itself.

qualified
The qualified name of a member is its simple name prefixed by the name of its enclosing class, as in `OUDialog.request`. The parts of such names are separated by dots. This means that different classes can have members with the same names – the qualified names allow them to be distinguished from one another.

raw collection *
A collection for which an element type is not specified. Java collections prior to Java 5 are raw collections and contain only references of class `Object`.

`Reader`
The base class of a group of classes including `FileReader` and `BufferedReader`, which are used to read 16-bit character streams. `Reader` classes are used to read text files.

receiver
The object to which a message is sent.

recursive method
A method that calls itself as part of its method definition. This can lead to indefinite looping when an attempt is made to execute the method. However, used properly, it can be an extremely powerful approach.

refactoring
Refactoring is when code is rewritten without changing its overall effect, but for the purpose of improving its design, removing code duplication, or improving maintainability.

reference type variable
A variable declared to hold a reference to an object of the declared type (or a compatible type). Less formally we refer to a reference variable.

relative pathname
A path to a file beginning from the current working directory, often resulting in a shorter path than an absolute path.

return type
The type of value returned by a method.

run-time
The period during which a program is executing, in contrast to the time period during which it is loaded or compiled.

run-time error
A programming error that becomes apparent only when the program is run, and has not been detected beforehand. Run-time errors can be caused by logical errors, failure to account for different ways in which code might be used, or conditions outside of the control of the programmer.

`Scanner`
A class used to read string tokens from a source. In M250 the source may be a `FileReader` or a `String`.

scope
The scope of a variable describes the areas of program code from which the variable may be used. The scope of a local variable is from the point where it is declared to the end of its enclosing statement block.

secondary sort
A sorting that can be applied after applying a primary sort in order to sort any elements that are considered equal by the primary sort.

selection
A technique by which a statement or statement block is executed subject to some Boolean condition. An `if` statement is used to perform selection in Java.

self-documenting code
Code written using well-chosen names and structures so as to make its intentions relatively clear when read.

sequence diagram
A diagram that depicts the interactions required between objects to carry out a particular task; this collaboration is shown in the form of messages and message answers.

set
A collection in which each item may appear only once – no duplicates are allowed.

setter message
A message that sets the value of one of a receiver's attributes. See also *getter message*.

setter method
A method whose purpose is to assign a new value to an instance variable. For example, the method `setPosition()` is a setter method for the instance variable `position` held by instances of the `Frog` class.

short-circuit evaluation
See *lazy evaluation*.

signature
See *method signature*.

software
A general term for all programs that can be run on a desktop computer or another hardware platform, such as a mobile phone.

software component
A piece of software that can be readily combined with other components to construct more complex software.

sorted collection
A collection that always keeps its elements in their natural ordering, if any. This contrasts with an ordered collection, where the ordering may just be an accident of where elements happen to have been placed in the collection.

source code
Program text expressed in a high-level programming language.

stable sort
A sort that does not reorder equal elements.

stack trace
A stack of textual information ordered from the last thing that happened down to the first thing that happened that is displayed when an exception is thrown and not handled; a representation of a call stack.

state
The values of the attributes of an object constitute its state. The state of an object can vary over time as the values of its attributes change.

statement
A statement represents a single instruction for the compiler to translate into bytecode. In Java most statements end with a semicolon.

statement block
A statement or sequence of statements 'bundled together' for use in a particular context. Any sequence of statements can be turned into a block by enclosing it in braces (curly brackets).

`static`
A Java keyword that defines a variable or method as belonging to a class rather than its instances.

static binding
The selection by the compiler at compile time (rather than a virtual machine at run-time) of a method to be executed at run-time. In the case of class methods, the compiler chooses the method from the class whose name is used to qualify the method name, or based on the type of the reference variable. In comparison, instance methods can be selected for execution at run-time based on the type of the receiver object. See *dynamic binding*.

static method
See *class method*.

static variable
See *class variable*.

stream
An object used to connect a data source to a data sink, enabling data to be transferred from the source to the sink.

string
A sequence of characters enclosed in quotation marks; an object of type `String`.

string interning
A process used by Java to create a pool of strings that can be reused when possible, which may result in the sharing of string literals.

`StringBuilder`
A class whose instances represent a mutable sequence of characters.

strong typing
A programming language feature that requires variables to have a declared type and enforces rules on the ways in which those types can be used, including what types of values can be assigned to variables of those types and what operators values of those types can be used with. Languages (such as Java) that feature strong typing are said to be strongly typed.

sub-array
A set of contiguous components within the same array.

subclass
A subclass is any class which extends (inherits from) another class. In Java all classes except `Object` are subclasses of some other class.

subclassing
Defining a class as a subclass of an existing class.

subinterface
A subinterface is an interface that extends another interface (called its superinterface), thereby inheriting the signatures of the parent interface, and which typically contains additional method signatures.

sublist
A list together with any two positions in the list may be seen as defining a new list containing just those elements between the two positions, retaining the same order. The new list is known as a sublist.

substitutability
The principle in object-oriented programming whereby, wherever the system expects an object of type `X`, an object of class `Y` can always be substituted instead, where class `Y` is a subclass of `X`, or where class `Y` implements an interface of type `X`. See *substitution*.

substitution
The technique of providing an instance of one class in a situation where an instance of a different class is expected. In Java, an object can be assigned to a variable whose type has been declared as some superclass of the object, or which has been declared as an interface type which that object's class implements. Similarly, an object can be used as an actual argument to a method where the formal argument's type has been declared as some superclass of the object, or which has been declared as an interface type which that object's class implements.

subtype
A type whose values (in the case of primitive types) or instances (in the case of class types) are substitutable for another type's values. For example, a subclass type is a subtype of a superclass type. A class that implements an interface is a subtype of the interface type.

`super`
A Java keyword that allows the writing of expressions to access members of a class that are inaccessible using the expression `this`, due to hiding or overriding; for example, `super.x` or `super.x()` causes the JVM to start its search for the corresponding member in the superclass of the receiver. The member must be accessible for such code to compile.

`super()`
Used within a constructor to invoke the constructor without any arguments in the direct superclass. May also be used with *n* arguments to invoke a superclass constructor with *n* formal arguments.

superclass
If `B` is a subclass of `A`, then `A` is the superclass of `B`. In the Java programming language a subclass has only one direct superclass.

superinterface
A superinterface is an interface that is extended by some other interface(s).

supertype
A type that supports substitution of more specific types. Both interface types and superclass types (including abstract class types) are examples of supertypes. See also *subtype*.

syntax
The syntax of a programming language is the set of rules governing the structure of the language, including its valid symbols, expressions and structures. Syntax rules define the form of the various constructs in the language, but say nothing about the meaning of these constructs.

syntax error
An error that occurs as a result of failure to observe a syntax rule. Such an error is detected during compilation by a compiler.

testing
The process of using test cases to check that code is behaving as the programmer intended, and to help discover bugs.

`this`
An expression used within a method or constructor to reference the object executing the method or constructor.

`this()`
Used within a constructor to invoke the constructor without any arguments in the current class. May also be used with *n* arguments to invoke a constructor with *n* formal arguments in the current class.

throw
The process of throwing an exception, also referred to as raising or signalling an exception.

`throws` **clause**
A part of a method header beginning with the keyword `throws`, followed by a comma-separated list of any exceptions the programmer wants to declare to be thrown. If a checked exception is not handled, it must be declared in a `throws` clause.

token
An item of data that can be written to or retrieved from a file. Delimiters are used to separate tokens in a file.

tree
A collection structure typified by branching connections between elements, often used to create collections of items sorted according to some criteria.

***try-catch* statement**
An exception handling statement, a mechanism to catch exceptions. If a possible checked exception is not declared in a `throws` clause, it must be handled using a *try-catch* statement.

unary operator
An operator that has just one operand; for example, the minus operator.

unchecked exception
An exception for which the compiler does not require any explicit exception handling code and that the compiler does not require the programmer to declare as thrown in a method header. These are errors that may not be easily recovered from, or should not be recovered from; instead, program logic or other issues may need to be resolved.

union
Union is a mathematical operation on two or more sets that results in a set containing all of the elements of all of the sets. The term is used to refer both to the result and to the operation.

unwrapping
The process of converting a wrapped value (one contained in a wrapper class) to its primitive value equivalent.

utility class *
A utility class is one that provides methods (usually static) that perform common functions. The `Math` class is an example.

variable
A named 'chunk' or block of the computer's memory which can hold either a value of some primitive type or a reference to an object.

variable initialisation
The process of assigning a variable a value for the first time.

variable reference diagram
A diagram showing the relationships between reference variables, objects and their instance variables. Object protocol is not included.

virtual machine (VM)
A layer of software that simulates a computer capable of interpreting bytecode.

visibility
Visibility of an item (such as a member or constructor) in a class refers to whether or not parts of a class can access that item directly. Same as *accessibility*.

`void`
A keyword used in Java to indicate that a method does not return a value.

`while` **statement**
A loop that allows a statement block to be repeatedly executed depending on the value of some Boolean condition. `while` loops are typically used to repeat code when the number of iterations cannot be determined in advance. See also *`for` loop*.

workspace variable
Workspace variables (such as in the OUWorkspace) behave like local variables, except that their lifetime lasts until you close or reset the workspace, rather than ending when a particular block has finished executing.

wrapper class
Every primitive type has a corresponding wrapper class whose purpose is to hold a primitive value in a place where a primitive value is intuitively what is needed, but where the type checking requires an object. For example, the wrapper class of `int` is `Integer`. Wrapper classes are used in auto-boxing.

wrapping
The process of converting a primitive value to an object equivalent, using a wrapper class.

`Writer`
The base class of a group of classes including `FileWriter` and `BufferedWriter`, which are used to write 16-bit character streams. `Writer` classes are used to write text files.

zero-based indexing
An indexing system in which the first index is `0`.