A

abstract class

A class with the `abstract` reserved word in its header. Abstract classes are distinguished by the fact that you may not *directly* construct objects from them using the `new` operator. An abstract class may have zero or more *abstract methods*.

abstract method

A method with the `abstract` reserved word in its header. An abstract method has no *method body*. Methods defined in an *interface* are always abstract. The body of an abstract method must be defined in a *sub class* of an *abstract class*, or the body of a class implementing an interface.

Abstract Windowing Toolkit

The Abstract Windowing Toolkit (AWT) provides a collection of classes that simplify the creation of applications with graphical user interfaces. These are to be found in the `java.awt` *packages*. Included are classes for windows, frames, buttons, menus, text areas, and so on. Related to the AWT classes are those for the *Swing* packages.

aggregation

A relationship in which an object contains one or more other subordinate objects as part of its state. The subordinate objects typically have no independent existence separate from their containing object. When the containing object has no further useful existence, neither do the subordinate objects. For instance, a gas station object might contain several pump objects. These pumps will only exist as long as the station does. Aggregation is also referred to as the *has-a relationship*, to distinguish it from the *is-a relationship*, which refers to *inheritance*.

anonymous class

A class created without a class name. Such a class will be a *sub class* or an implementation of an *interface*, and is usually created as an *actual argument* or returned as a method result. For instance

```
quitButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        System.exit(0);
    }
});
```

anonymous object

An object created without an *identifier*. They are usually created as array elements, *actual arguments* or method results. For instance

```
private Point[] vertices = {
    new Point(0,0),
    new Point(0,1),
    new Point(1,1),
    new Point(1,0),
};
```

See *anonymous class*, as these often result in the creation of anonymous objects.

append mode

A file writing mode, in which the existing contents of a file are retained when the file is opened. New contents are appended to the existing.

applet
: Applets are Java programs based around the `Applet` or `JApplet` classes. They are most closely associated with the ability to provide active content within Web pages. They have several features which distinguish them from ordinary Java graphical *applications*, such as their lack of a user-defined main method, and the security restrictions that limit their abilities to perform some normal tasks.

application
: Often used simply as a synonym for *program*. However, in Java, the term is particularly used of programs with a *Graphical User Interface (GUI)* that are not *applets*.

application programming interface (API)
: A set of definitions that you can make use of in writing programs. In the context of Java, these are the packages, classes, and interfaces that can be used to build complex applications without having to create everything from scratch.

B

base case
: A non-recursive route through a recursive method.

base type
: The type of items which may be stored in an *array* - the array's defined type. For instance, in
```
int[] numbers;
```
the base type of `numbers` is `int`. Where the base type is a class type, it indicates the lowest *super type* of objects that may be stored in the array. For instance, in
```
Ship[] berths;
```
only *instances* of the `Ship` class may be stored in `berths`. If the base type of an array is `Object`, instances of any class may be stored in it.

big-endian
: A common difference between machines is the order in which they store the individual bytes of multi-byte numerical data. A big-endian machine stores the higher-order bytes before the lower-order bytes. See *little-endian*.

binary search
: A search of sorted data, in which the middle position is examined first. Search continues with either the left or the right portion of the data, thus eliminating half the remaining search space. This process is repeated at each step, until either the required item is found, or there is no more data to search.

bit manipulation operator
: Operators, such as `&`, `|` and `^`, that are used to examine an manipulate individual *bits* within the bytes of a data item. The shift operators, `<<`, `>>` and `>>>`, are also bit manipulation operators.

boundary error
: Errors that arise from programming mistakes made at the edges of a problem - indexing off the edge of an array, dealing with no items of data, loop termination and so on. Boundary errors are a very common type of logical error.

bounds
: The limits of an *array* or collection. In Java, the lower limit is always zero. In the case of an array, the upper bound is one less than then length of the array, and is fixed.

Indexing outside the bounds of an array or collection will result in an `IndexOutOfBoundsException` *exception* being thrown.

bytecode
Java source files are translated by a *compiler* into bytecodes - the *instruction set* of the *Java Virtual Machine (JVM)*. Bytecodes are stored in `.class` files.

C

catch clause
The part of a *try statement* responsible for handling a caught *exception*.

catching exceptions
Exceptions are caught within the *catch clause* of a *try statement*. Catching an exception gives the program an opportunity to recover from the problem or attempt a repair for whatever caused it.

character set encoding
The set of values assigned to characters in a character set. Related characters are often grouped with consecutive values, such as the alphabetic characters and digits.

checked exception
An *exception* that must be caught locally in a *try statement*, or propagated via a *throws clause* defined in the *method header*. See *unchecked exception*.

command-line argument
Arguments passed to a program when it is run. A Java program receives these in the single formal argument to its *main method*
`public static void main(String[] args)`
The arguments are stored as individual strings.

compilation
The process of translating a programming language. This often involves translating a *high level programming language* into a *low level programming language*, or the binary form of a particular *instruction set*. The translation is performed by a program called a *compiler*. A Java compiler translates programs into *bytecodes*.

compiler
A program which performs a process of *compilation* on a program written in a *high level programming language*.

complement operator
The complement operator, `~`, is used to invert the value of each *bit* in a *binary* pattern. For instance, the complement of `1010010` is `0101101`.

concurrency
A feature of *parallel programming*. Parts of a program whose executions overlap in time are said to execute concurrently. Java's *thread* feature support concurrency.

coupling
Coupling arises when classes are aware of each of other because their instances must interact. Linkage between two classes that may be either strong or weak. Stronger coupling arises when one class has a detailed knowledge of the internal implementation of another, and is written to take advantage of that knowledge. So anything that has the potential to reduce the amount of inside knowledge will tend to weaken coupling. Hence, *information hiding* and *encapsulation*. Java's visibility levels - private, package, protected, public - progressively reveal detail to other classes, and so increase the potential for stronger coupling. Interfaces are one way to

reduce to reduce coupling - because you interact with a class via an abstract definition, rather than a concrete implementation.

critical section

A section of code in which there is potential for a *race hazard*. Critical sections make use of the `synchronized` methods or statements.

cursor

A visual representation of the current position of the mouse on the user's *virtual desktop*. Cursor shapes are often set to represent the current state of a program - using an hour glass shape to indicate that the user should wait - or to suggest the actions that are possible by clicking the mouse over some part of a user interface.

D

daemon thread

Daemon threads are non-user threads. They are typically used to carry out low-priority tasks that should not take priority over the main task of the program. They can be used to do useful work when all other user threads are blocked. The *garbage collector* is one example of a daemon thread.

deadlock

A situation that arises when two *threads* each acquires the lock to one of a set of resources that they both need.

deep copy

A copy of an object in which copies of all the object's sub-components are also made. The resulting object might, in effect, be a *clone* of the original. See *shallow copy* for an alternative.

delegation

The process by which an object passes on a message it has received to a sub-ordinate object. If *inheritance* is not available in a programming language, delegation is the most viable alternative for avoiding code duplication and promoting code reuse.

direct recursion

Recursion that results from a method calling itself.

downcast

A *cast* towards an object's dynamic type - that is, `down' the *inheritance hierarchy*. For instance

```
// Downcast from Object to String
String s = (String) o;
```

See *upcast*.

dynamic type

The dynamic type of an object is the name of the class used to construct it. See *static type*.

E

edit-compile-run cycle

A common part of the program development process. A source file is created initially and compiled. Syntax errors must be corrected in the editor before compiling it again. Once the program has been successfully compiled, it can be run. The program's execution might reveal logical errors, or the need for enhancements. A further edit-compile-run iteration is the result.

---

exception
: An object representing the occurrence of an exceptional circumstance - typically, something that has gone wrong in the smooth running of a program. Exception objects are created from *classes* that extend the `Throwable` class. See *checked exception* and *unchecked exception*.

exception handler
: The *try statement* acts as an exception handler - a place where *exception* objects are caught and dealt with.
.

F

factory pattern
: A *pattern* of class definition that is used as a generator of *instances* of other classes. Often used to create platform- or locale-specific implementations of *abstract classes* or *interfaces*. This reduces *coupling* between classes as it frees the factory's client from a need to know about particular implementations.

filter stream
: An input-output class that filters or manipulates its stream of input- or output-data in some way. Two examples are `DataInputStream` and `DataOutputStream`.

final class
: A class with the `final` reserved word in its header. A final class may not be extended by another class.

finalization
: Immediately before an object is garbage collected, its `finalize` method is called. This gives it the opportunity to free any resources it might be holding on to.

finally clause
: Part of a *try statement* that is always executed, either following the handling a caught *exception*, or normal termination of the *protected statements*.

final method
: A method with the `final` reserved word in its header. A final method may not be overridden by a method defined in a sub class.

fully qualified class name
: The name of a class, including any *package* name and enclosing class name. Given the following class outline

```
package oddments;

class Outer {
    public class Inner {
        ...
    }
    ...
}
```

The fully qualified name of `Inner` is
`oddments.Outer.Inner`

fully evaluating operator
: An operator that evaluates all of its arguments to produce a result. Standard *arithmetic operators*, such as `+`, are fully evaluating. In contrast, some *boolean operators*, such as `&&`, are *short-circuit operators*.

G

garbage collector
A *daemon thread* that recycles objects to which there are no extant references within a program.

Graphical User Interface
A Graphical User Interface (GUI) is part of a program that allows user interaction via graphical components, such as menus, buttons, text areas, etc. Interaction often involves use of a mouse.

H

has-a relationship
See *aggregation*.

I

implements clause
That part of of a *class header* that indicates which interfaces are implemented by the class. A class may implement more than one interface. See *multiple inheritance*.

indirect recursion
Recursion that results from method `Y` calling method `X`, when an existing call from `X` to `Y` is still in progress.

infinite recursion
Recursion that does not terminate. This can result from any of *direct recursion*, *indirect recursion* or *mutual recursion*. It is usually the result of a logical error, and can result in *stack overflow*.

initializer
A *block* defined at the outermost level of a class - similar to a method without a header. Initializer blocks are executed, in order, when an *instance* is created. They are executed before the *constructor* of the defining class, but after any *super class* constructor. They are one of the places in which *blank final variables* may be initialized.

inner class
A class defined inside an enclosing class or method. We use the term to refer to non-static *nested classes*.

interface inheritance
When a *class* implements an *interface*, an interface *inheritance* relationship exists between them. The class inherits no implementation from the interface, only method signatures and *static variables*. It is also possible for one interface to extend one or more interfaces. In Java, interface inheritance is the only form of *multiple inheritance*. See *class inheritance* for an alternative form of inheritance.

interpretational inner class
An *inner class* whose role is to provide a view or interpretation of data belong to its enclosing class, but independent of the data's actual representation.

interpreter
A program which executes a translated version of a source program by implementing a *virtual machine*. Interpreters typically simulate the actions of an idealized *Central Processing Unit*. An interpreter for Java must implement the *Java Virtual Machine*

(JVM) and executes the *bytecodes* produced by a Java *compiler*. The advantage of using an interpreter for Java is that it make the language more *portable* than if it were a fully compiled language. The bytecode version of a program produced by a Java compiler may be run on any interpreter implementing the JVM.

interprocess communication
The ability of two or more separate *processes* to communicate with one another.

is-a relationship
See *inheritance*.

iterator pattern
A common *pattern* in which the contents of a collection are iterated over in order. The Iterator pattern frees a client of the data from needing details of the how the data is stored. This pattern is supported by the `Iterator` and `ListIterator` interfaces.

L

layout manager
An object responsible for sharing the available space between multiple components within a graphical container.

left shift operator
The left shift operator (<<) is a *bit manipulation operator*. It moves the bits in its left operand zero or more places to the left, according to the value of its right operand. Zero bits are added to the right of the result.

little-endian
A common difference between machines is the order in which they store the individual bytes of multi-byte numerical data. A little-endian machine stores the lower-order bytes before the higher-order bytes. See *big-endian*.

livelock
A situation in which a *thread* waits to be notified of a condition but, on waking, finds that another thread has inverted the condition again. The first thread is forced to wait again. When this happens indefinitely, the thread is in livelock.

local inner class
An *inner class* defined within a method.

look-and-feel
The visual impression and interaction style provided by a user interface. This is predominantly the responsibility of the *window manager* (in collaboration with the underlying *operating system*) running on a particular computer. It refers to style of such things as window title bars, how windows are moved and resized, how different operations are performed via a mouse, and so on. It is preferable to have a consistent look and feel within a single user environment. However, some window managers do allow individual programs to present a different look and feel from the predominant style of the host environment. Java's Swing components support this idea by allowing an application to select a `pluggable look and feel' from those provided by a user interface manager. An application running in a Microsoft Windows environment could be made to look like one that normally runs in an X Windows environment, for instance. This allows an application to look similar on different platforms, but it can also lead to confusion for users.

M

---

manifest file
A file held in a *Java Archive (JAR) file*, detailing the contents of the archive.

marking interface
An *interface* with no *methods*.

member
The members of a class are *fields*, *methods* and *nested classes*.

memory leak
A situation in which memory that is no longer being used has not been returned to the pool of free memory. A *garbage collector* is designed to return unreferenced objects to the free memory pool in order to avoid memory leaks.

message passing
We characterize object interactions as message passing. A *client* object sends a message to a *server* object by invoking a method from the server's class. Arguments may be passed with the message, and a result returned by the server.

modal
A dialog is modal if its parent application is blocked from further activity until the dialog has completed. See *non-modal*.

model-view-controller pattern
A *pattern* in which the representation of data (the model) is kept separate from its visualization (the view) and kept separate from the elements of the system which control the application (the controller). Such decoupling makes it easier to change the underlying data representation, or provide multiple views, for instance.

module
A group of program components, typically with restricted visibility to program components in other modules. Java uses *packages* to implement this concept.

monitor
An object with one or more synchronized methods.

multiple inheritance
The ability of a class or interface to extend more than one class or interface. In Java, multiple inheritance is only available in the following circumstances

- An interface may extend more than one interface.
- A class may implement more than one interface.

Only *single inheritance* is possible for a class extending another class.

multiprogramming system
An operating system that is able to run multiple programs concurrently.

mutual recursion
Recursion that results from two methods calling each other recursively.

N

nested class
A class defined inside an enclosing class. See *inner class*.

non-modal
A dialog is non-modal if its parent application is not blocked from further activity while the dialog is being shown. See *modal*.

non-static nested class
See *inner class*.

O

object serialization
: The writing of an object's contents in such a way that its *state* can be restored, either at a later time, or within a different *process*. This can be used to store objects between runs of a program, or to transfer objects across a network, for instance.

out-of-bounds value
: A *redundant value* used to indicate that a different action from the norm is required at some point. The `read` method of `InputStream` returns `-1` to indicate that the end of a stream has been reached, for instance, instead of the normal positive byte-range value.

overriding for breadth
: A form of *method overriding* in which the sub class version of a method implements its own behavior within the context of the attributes and behavior of the sub class and then calls the super class version so that it can perform a similar task within the super class context.

overriding for chaining
: A form of *method overriding* in which the sub class version of a method checks to see whether it can respond to the message on its own and only calls the super class version of the method.

overriding for restriction
: A form of *method overriding* in which the sub class version of a method calls the super class version first of all and then uses or manipulates the result or effects of that call in some way.

P

package
: A named grouping of classes and interfaces that provides a package *namespace*. Classes, interfaces and class members without an explicit `public`, `protected` or `private` *access modifier* {access!modifier} have *package visibility*. Public classes and interfaces may be imported into other packages via an *import statement*.

package declaration
: A declaration used to name a *package*. This must be the first item in a source file, preceding any *import statements*. For instance,
```
package java.lang;
```

parallel programming
: A style of programming in which statements are not necessarily executed in an ordered sequence but in parallel. Parallel programming languages make it easier to create programs that are designed to be run on multi-processor hardware, for instance. Java's *thread* features support a degree of parallel programming.

peer
: A term used of the *Abstract Windowing Toolkit (AWT)* to refer to the underlying classes that provide the platform-specific implementation of component classes.

pipe
: A linkage between two program components. One component acts as a source of data, and writes into the pipe. A second components acts as a receiver (sink) for the data and reads from the pipe. See PipedInputStream and PipedOutputStream.

pixel
: A `picture element' - typically a colored dot on a screen.

---

polling
: The process of repeatedly testing until a condition becomes true. Polling can be inefficient if the time between tests is short compared with the time it will take for the condition to become true. A polling *thread* should sleep between consecutive tests in order to give other threads a chance to run. An alternative approach to polling is to arrange for an *interrupt* to be sent when the condition is true, or to use the `wait` and `notify` mechanism associated with threads.

popup menu
: A menu of actions that is normally not visible on the screen until a mouse button is clicked. Popup menus help to keep a user interface from becoming cluttered.

preempt
: The currently executing *thread* may be preempted, or forced to yield control, by a higher priority thread that becomes eligible to run during its *timeslice*.

priority level
: Each *thread* has a priority level, which indicates to the *scheduler* where it should be placed in the pecking order for being run. An eligible un-blocked thread with a particular priority will *always* be run before an eligible thread with a lower priority.

process
: An individual thread-of-control to which an execution *timeslice* is allocated by an *operating system*.

propagation
: If an *exception* is thrown within a method, and there is no appropriate *exception handler* within the method, the exception may be propagated to the caller of the method. For a *checked exception*, the method must contain a *throws clause* in its header. A throws clause is not necessary for an *unchecked exception* {exception!unchecked} to be propagated.

protected statement
: A statement within the *try clause* of a *try statement*.

Q

quantum
See *timeslice*.

R

radio buttons
A group of selectable components in which only one component may be selected. Selection of one of the group causes the previously selected component to be deselected.

race condition
See *race hazard*.

race hazard
A situation that arises between multiple threads sharing a resource. A race hazard arises when one thread's assumptions about the state of a resource are invalidated by the actions of another thread.

Reader class
A *sub class* of the `Reader` *abstract*, defined in the `java.io` *package*. Reader classes translate input from a host-dependent *character set encoding* into *Unicode*. See *Writer class*.

recursion
Recursion results from a method being invoked when an existing call to the same method has not yet returned. For instance

```
public static void countDown(int n){
    if(n >= 0){
        System.out.println(n);
        countDown(n-1);
    }
    // else - base case. End of recursion.
}
```

See *direct recursion*, *indirect recursion* and *mutual recursion* for the different forms this can take.

redundant value
The value of a data type that has no use or meaning within a particular context. For instance, negative values would be redundant a class using integer attributes to model assignment marks. In some applications, redundant patterns serve a useful purpose in that they can be used explicitly as *out-of-bounds values* or *escape values*.

reflection
The ability to find out what methods, fields, constructors, and so on, are defined for a class or object. Reflection is supported by the `Class` class, and other classes in the `java.lang.reflect` *package*. Reflection makes it possible, among other things, to create dynamic programs.

relational operators
Operators, such as `<`, `>`, `<=`, `>=`, `==` and `!=`, that produce a boolean result, as part of a *boolean expression*.

relative filename
A filename whose full path is relative to some point within a *file system* tree - often the current working folder (directory). For instance

---

`../bin/javac.exe`

A relative filename could refer to different files at different times, depending upon the context in which it is being used. See *absolute filename*.

right shift operator

The right shift operator (>>) is a *bit manipulation operator*. It moves the bits in its left operand zero or more places to the right, according to the value of its right operand. The most significant bit from before the shift is replicated in the leftmost position - this is called *sign extension*. An alternative right shift operator (>>>) replaces the lost bits with zeros at the left.

round robin allocation

An allocation of *timeslices* that repeatedly cycles around a set of eligible *threads* in a fixed order.

runtime stack

A stack structure maintained by the Java Virtual Machine that records which methods are currently being executed. The most recently entered method will be at the top of the stack and the main method of an application will be near the bottom.

S

search path

A list of folders (directories) to be searched - for a program or class, for instance.

shallow copy

A copy of an object in which copies of all the object's sub-components are not also made. For instance, a shallow copy of an array of objects would result in two separate array objects, each containing references to the same set of objects as were stored in the original. See *deep copy* for an alternative.

shift operator

See *left shift operator* and *right shift operator*.

shortcut key

A key-press associated with a component in a *Graphical User Interface (GUI)* that provides an alternative to selecting the component's operation with a mouse.

singleton pattern

A *pattern* that allows us to ensure that only a single instance of a particular class exists at any one time. Such an instance is called a singleton. The pattern can also be used when instances would have no unique state and would behave identically.

static nested class

A nested class with the `static` reserved word in its header. Unlike *inner classes*, objects of static nested classes have no enclosing object. They are also known as nested top-level classes.

static type

The static type of an object is the declared type of the variable used to refer to it. See *dynamic type*.

stream class

An input stream class is one that delivers data from its source (often the *file system* as a sequence of bytes. Similarly, an output stream class will write byte-level data. Stream classes should be contrasted with the operation of *reader* and *writer* classes.

subordinate inner class

An *inner class* that performs well-defined subordinate tasks on behalf of its *enclosing class*.

---

sub type
: A type with a parent *super type*. The sub-type/super-type relationship is more general than the sub-class/super-class relationship. A class that implements an *interface* is a sub type of the interface. An interface that extends another interface is also a sub type.

super type
: A type with a child *sub type*. The sub-type/super-type relationship is more general than the sub-class/super-class relationship. An interface that is implemented by a class is a super type of the class. An interface that is extended by another interface is also a super type.

Swing
: The Swing classes are part of a wider collection known as the *Java Foundation Classes (JFC)*. Swing classes are defined in the `javax.swing` packages. They provide a further set of components that extend the capabilities of the *Abstract Windowing Toolkit (AWT)*. Of particular significance is the greater control they provide over an application's *look-and-feel*.

swizzling
: The process of recursively writing the contents of an object via *object serialization*.

synchronized statement
: A statement in which an object-lock must be obtained for the target object before the body of the statement can be entered. Used to enclose a *critical section* in order to prevent a *race hazard*.

T

thread
: A lightweight *process* that is managed by the *Java Virtual Machine (JVM)*. Support for threads is provided by the `Thread` class in the `java.lang` package.

thread starvation
: A condition that applies to a *thread* that is prevented from running by other threads that do not yield or become blocked.

throw an exception
: When an exceptional circumstance arises in a program - often as a result of a *logical error*, and *exception* object is created and thrown. If the exception is not caught by an *exception handler*, the program will terminate with a *runtime error*.

throws clause
: A clause in a *method header* indicating that one or more *exceptions* will be *propagated* from this method. For instance
```
public int find(String s) throws NotFoundException
```

throw statement
: A statement used to *throw an exception*. For instance
```
throw new IndexOutOfBoundsException(i+" is too large.");
```

timesharing system
: An operating system that shares processor time between multiple processes by allocating each a *timeslice*. Once a process's timeslice has expired, another process is given a chance to run.

timeslice
: The amount of running time allocated to a *process* or *thread* before the *scheduler* considers another to be run. A process or thread will not be able to use its full allocation of time if it becomes blocked or *preempted* during this period.

---

top level class
A class defined either at the outermost level of a *package* or a *static nested class*.

trusted applet
An *applet* with more privileges than an ordinary (untrusted) applet.

try clause
See *try statement*.

try statement
The try statement acts as an *exception handler* - a place where *exception* objects are caught and dealt with. In its most general form, it consists of a *try clause*, one or more *catch clauses* and a *finally clause*.

```
try{
    statement;
    ...
}
catch(Exception e){
    statement;
    ...
}
finally{
    statement;
    ...
}
```

Either of the catch clause and finally clause may be omitted, but not both.

U

unchecked exception
An *exception* for which it is not required to provide a local *try statement*, or to propagate via a *throws clause* defined in the *method header*. An exception that is not handled will cause program termination if it is thrown. See *checked exception*.

unnamed package
All classes defined in files without a *package declaration* are placed in the unnamed package.

upcast
A cast towards an object's ultimate super type - that is, `up' the inheritance hierarchy towards the Object class, for instance

```
// Upcast from VariableController to HeaterController
VariableController v;
...
HeaterController c = v;
```

See *downcast*. Java's rules of *polymorphism* mean that an explicit upcast is not usually required.

W

Writer class
A *sub class* of the `Writer` *abstract*, defined in the `java.io` *package*. Writer classes translate output from *Unicode* to a host-dependent *character set encoding* . See *Reader class*.
