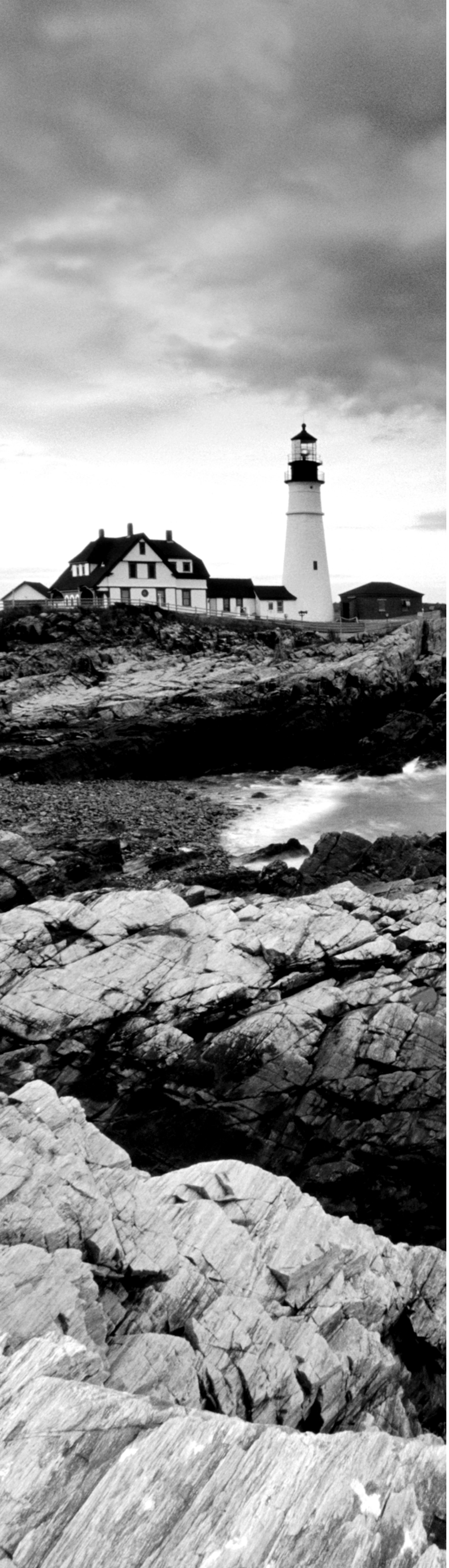

# Glossary

# A

**abstract class** A class that is marked with the abstract keyword and cannot be instantiated.

**abstract method** A method that is marked with the abstract keyword defined in an abstract class or interface, for which no implementation is provided in the class or interface in which it is declared. Any concrete subclass inheriting the abstract method must provide an implementation.

**accessor method** Method used to read an instance variable. Also known as a "getter."

**access modifier** Defines what other code can refer to the code. Choices are private, protected, public, and default (no modifier.)

**application programming interface (API)** Represents code made available for your program to call.

**arithmetic operator** Any unary or binary value that performs a mathematical calculation. Includes *, /, %, +, -, ++, and –.

**array** A contiguous chunk of memory on the heap representing a fixed-size collection of values that all have the same data type.

**assignment operator** An operator that assigns a value to a variable. Includes simple = and compound assignment operators: =, +=, -=, *=, /=, %=, &=, ^=, !=, <<=, >>=, and >>>=.

**autoboxing** The compiler automatically converts a primitive type into the matching wrapper class.

# B

**binary operator** An operator that takes two operands. Java includes a variety of binary operators, such as arithmetic, logical, bitwise, equality, relational, and assignment operators.

**bitwise operator** An operator that performs a bit comparison between two values. Includes: &, ^, and |.

**block** A grouping of code that is surrounded by {}.

**break statement** A statement within a switch, while, do-while, or for statement that causes the control flow to exit the structure and returns control to the enclosing statement.

**bytecode** Compiled Java code. Bytecode appears in files with the .class extension.

# C

**case statement** Defines an execution entry point for a particular value within a switch statement. There may be at most one case statement for a given value within a single switch statement.

**catch block** A block of code using the catch keyword that follows a try block. The catch block is often referred to as an exception handler since it can handle an exception and stop it from traversing further down the method call stack.

**checked exception** Any exception that is a subclass of java.lang.Exception but not a subclass of java.lang.RuntimeException. Checked exceptions must be handled or declared.

**child class** When referring to two classes that share inheritance, the class that extends from another class.

**class** A description of an object. A class in Java is defined in a source file with the .java extension and compiled into a file with the .class extension.

**class variable** A field within a class declared as static.

**classpath** The path on your filesystem where Java will search for files with the .class extension. The classpath can be defined by the CLASSPATH environment variable.

**compound assignment operator** An assignment operator that first applies a numeric, logical, or shift operation to a value before assigning it to a variable. Includes =, +=, -=, *=, /=, %=, &=, ^=, !=, <<=, >>=, and >>>=.

**concatenation** Two Strings being merged together. For example, "1" + "2" evaluates to "12".

**concrete class** The first non-abstract subclass that extends an abstract class. It is required to implement all inherited abstract methods, which have not been implemented in a parent class.

**conditional operator** The operator a ? b : c, where a is a boolean expression, b is the expression when a is evaluated to true, and c is the expression when a is evaluated to false.

**constants** Variables that do not change during the program. They use the final keyword.

**constructor** A special method within a class that gets invoked when an object is instantiated. A constructor must match the name of the class and cannot declare a return value. The purpose of a constructor is to initialize the fields of the object.

**continue statement** A statement within a `while`, `do-while`, or `for` loop that causes the flow to stop the execution of the current iteration and go on to the next.

**control flow statement** A Java statement that changes the order of execution of an application at runtime, such as those that repeat or branch to particular segments of code.

**covariant return type** When a method in a child class overrides a method in a parent class, the return type is permitted to be a subclass of the return type of the method defined in the parent class.

# D

**default constructor** A constructor without any parameters that is generated by the compiler when no constructors are coded.

**default method** A method defined in an interface with the `default` keyword in which a method body is permitted.

**default package** An unnamed package used when no package statement is present in the code.

**`default` statement** Execution entry point when no case statement exists that matches a particular value within a `switch` statement. There may be at most one `default` statement within a single `switch` statement.

**deferred execution** A situation in which there is a block of code that will be run later.

**`do-while` loop** A repetition control structure that is executed at least once.

# E

**`else` statement** Part of an `if-then` statement that defines a statement or block that is executed if the `boolean` in the `if-then` statement evaluates to `false`.

**encapsulation** Combining fields and methods together in a class such that the methods operate on the data, as opposed to users of the class accessing the fields directly.

**equality operator** The `==` operator that determines if two primitive numeric types, `boolean` values, or object references are the same.

**exception** An event that occurs during the execution of a program that disrupts the normal flow of control. In Java, an exception is an object that a method “throws” down the method call stack by handing it to the JVM.

# F

**field** Another name for an instance variable in a class.

**`finally` clause** A block of code that follows a **`try`** statement and executes after the **`try`** block regardless of whether an exception occurs within the **`try`** block.

**`for` loop** A repetition control structure that includes a structure initialization, termination, and increment section. Often used to iterate over lists of items.

**`for-each` loop** An enhanced form of a `for` loop that can be applied to Java built-in arrays and objects that implement `java.lang.Iterable`, in which the loop variable structure is built and assigned automatically by the compiler.

**functional programming** A style of programming where you declare what you want to do by writing expressions.

# G

**`generics`** Refers to the new Java feature added to Java 2 Platform, Standard Edition (J2SE) 5.0 that provides support for parameterized data types.

# H

**handle or declare rule** A rule enforced by the compiler that states if a statement throws a checked exception, it must attempt to catch the exception or declare the exception in the method declaration using the `throws` keyword.

**happy path** The path through the code where exceptions are not thrown.

**heap** Represents a large pool of unused memory allocated to your Java application. All objects in Java reside in the heap memory.

**hidden method** When a child class defines a static method with the same name and signature as a static method defined in the parent class.

# I

**identifier** The name of a variable, method, class, interface, or enum.

**`if-then` statement** Simple form of decision control in Java in which a statement or block is executed if a `boolean` value evaluates to `true`.

**immutable** An object that cannot be changed.

**`import`** The Java keyword used to import a package into a source file.

**inheritance** The process by which the new child subclass automatically includes any public or protected primitives, objects, or methods defined in the parent class.

**initialization block** The section of a basic `for` loop that is executed at the start of the loop, before the first execution of the `boolean` expression. Variables declared within the initialization block of the `for` loop have limited scope and are not accessible outside the `for` loop.

**instance variable** The nonstatic fields in a class.

**instantiation** The process of bringing an `Object` to life. Includes calling the constructor.

**interface** An abstract data type that defines a list of abstract public methods that any class implementing the interface must provide. An interface can also include a list of constant variables and default methods.

# J

**Java Virtual Machine** Runs Java class files.

**JavaBean** A software component that makes it easy to reuse code.

# K

**keyword** Special words used by the compiler to indicate code constructs.

# L

**label** A pointer to the head of a statement that allows the application flow to jump to it or `break` from it. A label is a single-word expression followed by a colon (`:`).

**lambda expression** A block of code that can be passed around. Also known as a lambda or closure.

**literal** A numeric, character, `boolean`, or `String` value that is typed into the code.

**local variable** A variable defined within a method, including method parameters.

**logical complement operator** Given a `boolean` expression, the operator `!` returns the reverse of it.

**logical operator** An operator that compares two logical values. Includes: &, ^, |, &&, and ||.

**loop** A control flow statement that repeats a particular line or block of code.

# M

**member** A field or method in a class.

**method** Also known as a function—code that can be run by name.

**method declaration** The definition of a Java method consisting of seven components of a method: access modifier, optional specifiers, return type, method name, parameter list, exception list, and method body.

**method overloading** When a class contains multiple methods with the same name but different parameter lists.

**method signature** A method's name and parameter types.

**multiple inheritance** The rule a class may inherit from multiple direct parent classes. This feature is not supported in Java.

**mutable** An object that can change state.

**mutator method** Method used to change the value of an instance variable. Also known as a "setter."

# N

**negation operator** Reverses the sign of a numeric expression.

**numeric promotion** The process by which Java will automatically upcast the data type of a smaller data type to a larger data type, as well as update an integral value to a floating-point number.

# O

**object** An instance of a class.

**operator** A symbol in Java that is applied to an expression and returns a value.

**operator precedence** The order in which operators are evaluated.

**order of operation** *See* operator precedence.

**overflow** When a number value is too large to fit in a data type and the value automatically wraps around to the lowest possible value, and counts up from there.

**overriding** The classification for a new method in a class that has the same signature and return type as the method in a parent class.

# P

**package** A grouping of classes, interfaces, enumerations, and annotated types.

**parameter** The name of the variable in the method signature that is assigned the value of the argument.

**parent class** When referring to two classes that share inheritance, the class that is extended by another class.

**pointer** Java doesn't have pointers. A reference variable referring to an `Object` is as close as Java gets.

**polymorphic parameters** The property of method parameters in Java used to accept subclasses and subtypes of the method parameter type automatically.

**polymorphism** The property of a class in Java used to take on many different forms.

**post-decrement operator** The `--` operator when it is applied after a numeric variable. Decrements the value of the numeric variable by 1 and returns the original value.

**post-increment operator** The ++ operator when it is applied after a numeric variable. Increments the value of the numeric variable by 1 and returns the original value.

**pre-decrement operator** The `--` operator when it is applied before a numeric variable. Decrements the value of the numeric variable by 1 and returns the new value.

**pre-increment operator** The ++ operator when it is applied before a numeric variable. Increments the value of the numeric variable by 1 and returns the new value.

**primitive types** The built-in data types of the Java language. There are eight primitive types in Java: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, and `boolean`.

**property** What a JavaBean calls an instance variable.

# R

**reference types** Variables that are class types, interface types, and array types.

**reference variables** Variables that point to an object.

**relational operator** An operator that compares two expressions. Includes <, >, <=, >=, and instanceof.

**reserved word** Special words that may not be used as identifier names.

**return type** A primitive, reference type or the keyword void. Defines what, if anything, is sent back from the method to the caller.

**runtime exception** Any exception that is a subclass of java.lang.RuntimeException. The handle or declare rule does not apply to runtime exceptions.

# S

**scope** The portion of code where a variable can be accessed.

**short-circuit operator** A logical operator that may skip evaluation of the right-hand side of the expression if the value can be determined solely by the left-hand side of the equation. Includes && and ||.

**single inheritance** The rule that a class may inherit from only one direct parent class. Java supports single inheritance although the parent class may, in turn, inherit from another class. Java also supports implementing any number of interfaces.

**statement** A complete unit of execution in Java, terminated with a semicolon (;).

**static** A keyword indicating the variable or method is shared by all instances of a class.

**string** A sequence of characters.

**string pool** An area in the JVM where string literals are stored. The JVM can optimize the use of string literals by allowing only one instance of a string in the pool.

**switch statement** A complex decision-making structure in which a single value is evaluated and control jumps to a particular case statement or default block and continues until a break statement is reached or the switch statement finishes. Switch values may be bytes, shorts, chars, ints, enums, and Strings, as well as classes that wrap primitive data types.

# T

**ternary operator** *See* conditional operator.

**try statement** A block of code containing one or more statements that may throw an exception.

# U

**unary operator** An operator that takes only one operand. Includes `+`, `-`, `++`, `--`, and `!`.

**underflow** When a number value is too small to fit in a data type and the value automatically wraps around to the highest possible value, and counts down from there.

**update statement** The section of a basic `for` loop that is executed at the end of the loop, before the `boolean` expression is evaluated again.

# V

**vararg** A parameter that contains the ellipsis (...) after its data type can take in any number of arguments. This comma-separated list of arguments must appear at the end of the argument list and is treated as an array.

**variable** An allocated piece of memory for storing data. A variable has an identifier and a specific data type.

**virtual method** A method whose precise implementation is not determined until runtime. All nonfinal, nonprivate methods are virtual methods in Java since they may be overridden at runtime.

**`void`** Used in lieu of a type when a method does not return anything.

# W

**`while` loop** A repetition control structure that will continue until an expression evaluates to `true` or the loop is exited.