Weiss Chapter 1 terminology (parenthesized numbers are page numbers)

assignment operators In Java, used to alter the value of a variable. These operators include =, +=, -=, *=, and /=. (9)
autoincrement (++) **and autodecrement** (--) **operators** Operators that add and subtract 1, respectively. There are two forms of incrementing and decrementing prefix and postfix. (10)
binary arithmetic operators Used to perform basic arithmetic. Java provides several, including +, -, *, /, and %. (10)
block A sequence of statements within braces. (13)
break statement A statement that exits the innermost loop or switch statement. (16)
bytecode Portable intermediate code generated by the Java compiler. (4)
call-by-value The Java parameter-passing mechanism whereby the actual argument is copied into the formal parameter. (18)
comments Make code easier for humans to read but have no semantic meaning. Java has three forms of comments. (5)
conditional operator (?:) An operator that is used in an expression as a short-hand for simple if-else statements. (17)
continue statement A statement that goes to the next iteration of the innermost loop. (17)
do statement A looping construct that guarantees the loop is executed at least once. (15)
equality operators In Java, == and != are used to compare two values; they return either true or false (as appropriate). (11)
escape sequence Used to represent certain character constants. (7)
for statement A looping construct used primarily for simple iteration. (14)
identifier Used to name a variable or method. (7)
if statement The fundamental decision maker. (13)
integral types byte, char, short, int, and long. (6)
java The Java interpreter, which processes bytecodes. (4)
javac The Java compiler; generates bytecodes. (4)
labeled break statement A break statement used to exit from nested loops. (16)
logical operators &&, ||, and !, used to simulate the Boolean algebra concepts of AND, OR, and NOT. (12)
main The special method that is invoked when the program is run. (6)
method The Java equivalent of a function. (18)
method declaration Consists of the method header and body. (18)
method header Consists of the name, return type, and parameter list. (18)
null statement A statement that consists of a semicolon by itself. (13)
octal and hexadecimal integer constants Integer constants can be represented in either decimal, octal, or hexadecimal notation. Octal notation is indicated by a leading 0; hexadecimal is indicated by a leading 0x or 0X. (7)
overloading of a method name The action of allowing several methods to have the same name as long as their parameter list types differ. (19)
primitive types In Java, integer, floating-point, Boolean, and character. (6)
relational operators In Java, <, <=, >, and >= are used to decide which of two values is smaller or larger; they return true or false. (11)

return statement A statement used to return information to the caller. (19)
short-circuit evaluation The process whereby if the result of a logical operator can be determined by examining the first expression, then the second expression is not evaluated. (12)
signature The combination of the method name and the parameter list types. The return type is not part of the signature. (18)
standard input The terminal, unless redirected. There are also streams for standard output and standard error. (4)
static final entity A global constant. (20)
static method Occasionally used to mimic C-style functions; discussed more fully in Section 3.6. (18)
string constant A constant that consists of a sequence of characters enclosed by double quotes. (7)
switch statement A statement used to select among several small integral values. (17)
type conversion operator An operator used to generate an unnamed temporary variable of a new type. (10)
unary operators Require one operand. Several unary operators are defined, including unary minus (-) and the autoincrement and autodecrement operators (++ and --). (10)
Unicode International character set that contains over 30,000 distinct characters covering the principle written languages. (6)
while statement The most basic form of looping. (14)
Virtual Machine The bytecode interpreter. (4)

Weiss Chapter 2 terminology

aggregate A collection of objects stored in one unit. (37)
array Stores a collection of identically typed objects. (37)
array indexing operator [] Provides access to any element in the array. (37)
ArrayList Stores a collection of objects in array-like format, with easy expansion via the add method. (42)
call-by-reference In many programming languages, means that the formal parameter is a reference to the actual argument. This is the natural effect achieved in Java when call-by-value is used on reference types. (33)
catch block Used to process an exception. (48)
checked exception Must be either caught or explicitly allowed to propagate by a throws clause. (50)
command-line argument Accessed by a parameter to main. (45)
construction For objects, is performed via the new keyword. (31)
dot member operator (.) Allows access to each member of an object. (30)
dynamic array expansion Allows us to make arrays larger if needed. (40)
enhanced *for* loop Added in Java 5, allows iteration through a collection of items. (46)
equals Used to test if the values stored in two objects are the same. (34)
Error An unrecoverable exception. (50)
exception Used to handle exception occurrences, such as errors. (47)
FileReader Used for file input. (56)

FileWriter Used for file output. (59)
finally clause Always executed prior to exiting a try/ catch sequence. (48)
garbage collection Automatic reclaiming of unreferenced memory. (31)
immutable Object whose state cannot change. Specifically, Strings are immutable. (35)
input and **output** (I/ O) Achieved through the use of the java. io package. (51)
java.io Package that is used for nontrivial I/ O. (51)
length field Used to determine the size of an array. (38)
length method Used to determine the length of a string. (36)
lhs and **rhs** Stand for left-hand side and right-hand side, respectively. (32)
multidimensional array An array that is accessed by more than one index. (45)
new Used to construct an object. (31)
null reference The value of an object reference that does not refer to any object. (28)
NullPointerException Generated when attempting to apply a method to a null reference. (31)
object A nonprimitive entity. (30)
reference type Any type that is not a primitive type. (30)
runtime exception Does not have to be handled. Examples include **ArithmeticException** and NullPointerException. (49)
Scanner Used for line-at-a-time input. Also used to extract lines, strings, and primitive types from a single character source such as an input stream or String. Found in the java. util package. (52, 53)
String A special object used to store a collection of characters. (35)
string concatenation Performed with + and += operators. (35)
System. in, **System. out**, and **System. err** The predefined I/ O streams. (53)
throw clause Used to throw an exception. (51)
throws clause Indicates that a method might propagate an exception. (51)
toString method Converts a primitive type or object to a String. (37)
try block Encloses code that might generate an exception. (48)

Weiss Chapter 3 terminology

accessor A method that examines an object but does not change its state. (76)
aliasing A special case that occurs when the same object appears in more than one role. (81)
atomic unit In reference to an object, its parts cannot be dissected by the general users of the object. (70)
class Consists of fields and methods that are applied to instances of the class. (71)
class specification Describes the functionality, but not the implementation. (73)
CLASSPATH variable Specifies directories and files that should be searched to find classes. (94)
composite pattern The pattern in which we store two or more objects in one entity. (96)
constructor Tells how an object is declared and initialized. The default constructor is a member-by-member default initialization, with primitive fields initialized to zero and reference fields initialized to null. (76)

design pattern Describes a problem that occurs over and over in software engineering, and then describes the solution in a sufficiently generic manner as to be applicable in a wide variety of contexts. (96)
encapsulation The grouping of data and the operations that apply to them to form an aggregate while hiding the implementation of the aggregate. (70)
equals method Can be implemented to test if two objects represent the same value. The formal parameter is always of type Object. (78)
field A class member that stores data. (72)
implementation Represents the internals of how the specifications are met. As far as the class user is concerned, the implementation is not important. (73)
import directive Used to provide a shorthand for a fully qualified class name. Java 5 adds the static import that allows a shorthand for a static member. (91)
information hiding Makes implementation details, including components of an object, inaccessible. (70)
instance members Members declared without the static modifier. (83)
instanceof operator Tests if an expression is an instance of a class. (82)
javadoc Automatically generates documentation for classes. (73)
javadoc tag Includes @ author, @ param, @ return, and @ throws. Used inside of javadoc comments. (74)
method A function supplied as a member that, if not static, operates on an instance of the class. (71)
mutator A method that changes the state of the object. (76)
object An entity that has structure and state and defines operations that may access or manipulate that state. An instance of a class. (70)
object-based programming Uses the encapsulation and information-hiding features of objects but does not use inheritance. (71)
object-oriented programming Distinguished from object-based programming by the use of inheritance to form hierarchies of classes. (69)
package Used to organize a collection of classes. (90)
package statement Indicates that a class is a member of a package. Must precede the class definition. (93)
package-visible access Members that have no visibility modifiers are only accessible to methods in classes in the same package. (95)
package-visible class A class that is not public and is accessible only to other classes in the same package. (95)
pair The composite pattern with two objects. (96)
private A member that is not visible to nonclass methods. (72)
public A member that is visible to nonclass methods. (72)
static field A field that is shared by all instances of a class. (83)
static initializer A block of code that is used to initialize static fields. (86)
static method A method that has no implicit this reference and thus can be invoked without a controlling object reference. (83)
this constructor call Used to make a call to another constructor in the same class. (**82**)
this reference A reference to the current object. It can be used to send the current object, as a unit, to some other method. (79)
toString method Returns a String based on the object state. (78)

Weiss Chapter 4 terminology

abstract class A class that cannot be constructed but serves to specify functionality of derived classes. (129)
abstract method A method that has no meaningful definition and is thus always defined in the derived class. (128)
adapter class A class that is typically used when the interface of another class is not exactly what is needed. The adapter provides a wrapping effect, while changing the interface. (143)
anonymous class A class that has no name and is useful for implementing short function objects. (163)
base class The class on which the inheritance is based. (114)
boxing Creating an instance of a wrapper class to store a primitive type. In Java 5, this is done automatically. (145)
composition Preferred mechanism to inheritance when an IS-A relationship does not hold. Instead, we say that an object of class B is composed of an object of class A (and other objects). (110)
covariant arrays In Java, arrays are covariant, meaning that Derived[] is type compatible with Base[]. (124)
covariant return type Overriding the return type with a subtype. This is allowed starting in Java 5. (124)
decorator pattern The pattern that involves the combining of several wrappers in order to add functionality. (141)
derived class A completely new class that nonetheless has some compatibility with the class from which it was derived. (114)
dynamic dispatch A runtime decision to apply the method corresponding to the actual referenced object. (117)
extends clause A clause used to declare that a new class is a subclass of another class. (114)
final class A class that may not be extended. (120)
final method A method that may not be overridden and is invariant over the **inheritance hierarchy**. Static binding is used for final methods. (119)
function **object** An object passed to a generic function with the intention of having its **single** method used by the generic function. (158)
functor A function object. (158)
generic classes Added in Java 5, allows classes to specify type parameters and avoid significant amounts of type casting. (150)
generic programming Used to implement type independent logic. (142)
HAS-A relationship A relationship in which the derived class has a (instance of the) base class. (110)
implements clause A clause used to declare that a class implements the methods of an interface. (135)
inheritance The process whereby we may derive a class from a base class without disturbing the implementation of the base class. Also allows the design of class hierarchies, such as Throwable and InputStream. (114)
interface A special kind of abstract class that contains no implementation details. (134)

IS-A relationship A relationship in which the derived class is a (variation of the) base class. (117)
leaf class A final class. (120)
local class A class inside a method, declared with no visibility modifier. (161)
multiple inheritance The process of deriving a class from several base classes. Multiple inheritance is not allowed in Java. However, the alternative, multiple interfaces, is allowed. (131)
nested class A class inside a class, declared with the static modifier. (161)
partial overriding The act of augmenting a base class method to perform additional, but not entirely different, tasks. (121)
polymorphism The ability of a reference variable to reference objects of several different types. When operations are applied to the variable, the operation that is appropriate to the actual referenced object is automatically selected. (116)
protected class member Accessible by the derived class and classes in the same package. (118)
raw class A class with the generic type parameters removed. (154)
static binding The decision on which class's version of a method to use is made at **compile time**. Is only used for static, final, or private methods. (120)
static overloading The first step for deducing the method that will be used. In this step, the static types of the parameters are used to deduce the signature of the method that will be invoked. Static overloading is always used. (165)
subclass/**superclass** relationships If X IS-A Y, then X is a subclass of Y and Y is a **superclass of X.** These relationships are transitive. (117)
super constructor call A call to the base class constructor. (119)
super object An object used in partial overriding to apply a base class method. (121)
type bounds Specifies properties that type parameters must satisfy. (152)
type erasure The process by which generic classes are rewritten as nongeneric classes. (150)
type parameters The parameters enclosed in angle brackets <> in a generic class or method declaration. (150)
unboxing Creating a primitive type from an instance of a wrapper class. In Java 5, this is done automatically. (145)
wildcard types A ? as a type parameter; allows any type (possibly with bounds). (152)
wrapper A class that is used to store another type, and add operations that the primitive type either does not support or does not support correctly. (143)